Analysis of Implementations and Side-Channel Security of Frodo on Embedded Devices

By

MARCO MARTINOLI

Department of Computer Science
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in the Faculty of Engineering.

JANUARY 2020

Word count: 50,211

Abstract

Post-quantum cryptography is undergoing an in-depth scrutiny to assess practical aspects such as implementation performance and security in anticipation of its widespread future adoption. Frodo is one such cryptographic scheme, submitted to the NIST post-quantum standardisation effort. In this context, my contribution is twofold.

First of all, I apply several side-channel techniques to attack Frodo on a (emulated) ARM Cortex-M0. By using a single power consumption trace of a matrix multiplication involving secret material, I show how a divide-and-conquer technique can be used to mount an efficient key recovery attack, which however does not fully exploit the available leakage. Divide-and-conquer indeed assumes that leakage is independent across different subkeys, which is a limitation I overcome by mounting an extend-and-prune attack that exploits previously recovered subkeys to formulate an educated guess on intermediate variables. My study proceeds with the analysis of countermeasures: I show a deterministic countermeasure aimed at thwarting the extend-and-prune attack, I present a countermeasure that masks the Hamming weight thanks to the fact that secret elements are much smaller than the size of the space they live in, and finally I show how well-known countermeasures, such as blinding and masking, can be integrated into Frodo and assess the corresponding overhead.

My second contribution is a detailed analysis of the performances of Frodo on another embedded device, the ARM Cortex-M4. Although more powerful than the M0, this is still a very constrained environment where not all the matrices needed in the computations can be fully stored in memory, as they are too large. On-the-fly generation of such matrices is therefore required. I take the optimisations a step further by utilising ARM assembly instructions to multiply and accumulate 16-bit values as halfwords of 32-bit registers. Finally, I challenge the need for cryptographically secure PRNGs for the generation of public matrices in favour of faster non-cryptographic PRNGs. The result is a dramatic improvement in performance accompanied by an educated discussion about whether doing so affects security.

Dedication and acknowledgements

One of the best part of doing this PhD was meeting so many wonderful people. So many of them would deserve to be mentioned here, for helping me out with my work, for supporting me along the way, for sharing tough and hilarious moments with me or simply for grabbing a pint of warm English beer in a sticky Bristolian pub. I believe, one day, I'll thank the very last of you in person. For the time being, however, I definitely wish to start from this page.

Thanks to all my colleagues and co-authors for sharing with me an immense variety of moments: from incredibly deep conversations down the rabbit hole of Mathematics up to Brexit jokes. Those will never get old! I feel like an honourable mention is in order: endless unexpected discussions have enriched me and showed me that, no matter what, a new way of looking at things always exists. Thanks Dragoş!

Thanks to my explosive and extremely supportive supervisors, Elisabeth and Martijn: I will hold tight to your unique mix of methodical rigour and mad creativity. The patience you demonstrated in guiding me is definitely part of the reason why I got to write these lines.

My PhD wouldn't have been the same without the group of crazy folks I shared so many experiences with: ECRYPT-NET guys, you are awesome! I really hope our bond never vanishes and that, wherever life will bring us, the adventurous and joyous spirit with which we lived through our PhDs sparkles forever.

This happy life parenthesis as a whole would have never been possible without the help and support of my parents and family, Patrizia, Luciano, Lorenzo, Jacopo. I owe the person I am to you and your teachings: there aren't nearly enough words, in this or any other language, I could spend here to thank you enough. Therefore I won't. I will thank you day by day, with the simple gestures you taught me to change the world with.

Not a single person has made more efforts and sacrifices in the last eight years for me than you did. Your answer to my wish to start this adventure was "Awesome, when should I come?". And there you were, on the 14th of February 2016 after your one-way flight to London. You had the guts to leave everything behind just to follow someone else's dream, you fought through terrible challenges just to stick with me, out of love. I am not sure how many lifetimes I need to repay you for what you did, but I am sure I will start with this one. Thank you Ale, my wonderful and lovely wife.

Tutto il sudore e le lacrime versate le dedico a te, Nonna, perché non sono stati abbastanza per poterti far leggere queste parole di persona.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: MARCO MARTINOLI DATE: 31/01/2020

Table of Contents

List of Tables

List of Figures

FIGURE **Page**

Glossary

lattice mathematical structure defined as an additive discrete subgroup of the real numbers.

lattice-based cryptography branch of cryptography that studies schemes whose security is based on mathematical problems defined over lattices.

(Ring-/Module-)LWE (Ring-/Module-)Learning With Errors.

SVP Shortest Vector Problem.

CVP Closest Vector Problem.

BDD Bounded Distance Decoding.

sampler algorithm to sample a random variable according to a certain distribution.

side-channel measurable physical quantity from which it is possible to derive information, e.g. power consumed, time taken, EM radiated.

leakage information about sensitive values of an algorithm that is derived from side-channels.

side-channel analysis branch of cryptography that studies attacks that exploit side-channel information to retrieve secrets.

distinguisher mathematical function that discriminates between two distributions.

countermeasure alternative implementation of an algorithm that aims at reducing the information leaked through a side-channel.

divide-and-conquer side-channel attack strategy according to which portions of the secret (e.g. subkeys) are targeted independently from each other.

extend-and-prune side-channel attack strategy according to which portions of the secret (e.g. subkeys) are targeted sequentially in such a way that information derived on previous portions can be used to attack subsequent portions.

signal-to-noise ratio ratio between the amount of exploitable information an attacker can theoretically derive from a side-channel and independent noise.

power model formulated mathematical approximation of how a side-channel varies depending on the data being processed and on the operation being performed on a device.

template empirically derived approximate distributions of how a side-channel varies depending on the data being processed and on the operation being performed on a device.

algorithmic variance component of a side-channel that depends on the data being processed which is part of the targeted algorithm but outside the scope of an attack, and therefore is considered noise.

ELMO Emulator for power Leakages for the M0.

PRNG Pseudo Random Number Generator.

KEM Key Encapsulation Mechanism.

DEM Data Encapsulation Mechanism.

KEP Key Exchange Protocol.

PKC Public Key Cryptography.

DLP Discrete Logarithm Problem.

SIMD Single Instruction Multiple Data.

NIST National Institute of Standard and Technology.

Chapter 1

Introduction

I tend not to believe the weather forecast of more than a week, as it turns out to be too inaccurate for any meaningful planning. Scepticism is scientifically backed up by Lorenz [Lor63], who studied how systems of deterministic ordinary non-linear differential equations with bounded solutions are unstable. In other words, very small differences in initial conditions trigger considerably different results over time. Such a phenomenon later took the name of *the butterfly effect*, following the picturesque image of a butterfly causing a tornado on the other side of the world simply by flapping its wings.

The metaphor to which the butterfly effect owes its name is certainly an exaggeration, but the concept is well suited to describe how my PhD, and consequently this thesis, came to be. In 1980, the butterfly flapped its wings when Benioff [Ben80] constructed a quantum mechanical model of computation. He showed how quantum mechanical states could represent information and how to compute over such information by manipulating the states so to achieve the equivalent of a classical Turing machine.

His timing could not have been better as Manin [Man80] and later Feynman [Fey82] realised that quantum mechanics cannot be fully simulated on classical computers in an efficient manner. Feynman [Fey82] therefore brought forward the idea that a computer based on quantum states could be used to simulate quantum mechanics. He later refined Benioff's theoretical model [Ben80] in an attempt to substantiate his idea [Fey86]. Once the computational model was established, it appeared that there were applications other than simulating quantum mechanics for which quantum computers could be beneficial, i.e. problems they could solve significantly faster than any classical computer.

A long line of research looking for problems that could be efficiently solved on quantum computers was ignited. Initially, problems were tailor-made and found little prac-

tical application besides showing the potential of quantum computation. Each new result, however, inspired new and more sophisticated quantum algorithms solving more and more interesting problems, until a milestone was reached. In 1994, Shor [Sho94] developed an algorithm that leveraged a quantum subroutine to efficiently find the unknown period of periodic functions. As he himself showed, such a functionality can be relied upon to factor large integers or to compute discrete logarithms in (quantum) polynomial time, as opposed to the exponential time required by classical computers.

1.1 Motivations

Quantum computing is not the only branch of computer science which has influenced my work and, consequently, my thesis. In fact, my whole PhD can be thought of sitting at the intersection of two sub-areas of cryptography: post-quantum cryptography and side-channel analysis.

1.1.1 The impact of quantum computers on cryptography

The reason why Shor's algorithm is so ground-breaking is that the security of several widespread cryptographic schemes relies on the difficulty of solving those problems, most notably RSA [RSA78] and Diffie-Hellman [DH76]. Therefore, if a quantum computer powerful enough to implement Shor's algorithm with the correct parameters existed, all schemes of the above kind would be insecure. On the one hand, building quantum computers is still a hard engineering problem [DS13], most notably in how to deal with error propagation during the delicate steps required to carry out computations [KBF+15]. On the other hand, there are several good reasons to start preparing for a world where the strength of quantum computers could be harnessed by powerful adversaries to break current cryptoschemes. Such a world would need algorithms withstanding the known speed-ups offered by quantum computers: it would need *post-quantum cryptosystems*.

From a security point of view, a very good reason why initiating the post-quantum landscape does not have to wait for an actual full-scale quantum computer to be around is called *forward secrecy*. This is a security notion which was originally associated with key-agreement and key-exchange protocols and states that secrecy of session keys should not be compromised by future disclosure of the private key. In other words, if the private key is compromised at any given time, all past communication is still secure

[Gün90]. A very similar concept is pertinent to the post-quantum branch and is obtained by rephrasing the above notion: it is desirable that all present communications remain secure even if at some point in the future a quantum computer will be available. Clearly, none of the current communication infrastructures meet the forward secrecy property unless post-quantum algorithms are used. A very motivated adversary (organisation, state, etc.) could store all ciphertexts protected with classical cryptographic techniques and wait for a strong enough quantum computer to be available to decrypt them all.

1.1.2 The security and performance milestones

Mathematical notions of security model adversaries as having certain "powers" defined by the number of oracles they have access to when attacking a cryptosystem. The underlying assumption which is common to many notions is that, besides said oracles, an adversary has no control over the internals of the algorithm under attack. Such a model was challenged by Kocher et al. [Koc96, KJJ99], who showed how those kind of mathematical notions of security failed to capture many real-world situations: whenever a cryptosystem was implemented on a real device, it would inevitably offer to any adversary some information about the internal values of the algorithm. This could be leveraged to mount successful attacks, *side-channel attacks*, against the implementation. Some examples of such side-channels are:

- a hardware device consumes power in order to work. Obviously, this can be measured;
- due to basic physical principles of every electronic system, an electromagnetic field is released and can be measured;
- depending on the complexity of its tasks, a device can take a variable amount of time to perform them;
- in some very specific cases, with mechanical components involved, even sound information can be influenced by activities of the device.

These facts might sound fairly obvious at first, but the crucial realisation of Kocher et al. was that side-channels depend on the value of internal variables and on the operations being performed. If such a dependency is on secret material, it can be used to retrieve it.

Since the seminal publications by Kocher et al. [Koc96, KJJ99], much ground has been covered. Side-channel attacks have been proven successful against a wide range of cryptosystems, implemented on the most diverse underlying platforms. Techniques to attack and defend on the side-channel battlefield have been devised. For these reasons, it is natural to assume that side-channel techniques would still be relevant in the domain of post-quantum cryptography.

This is *the* reason why post-quantum algorithms have to be scrutinised through this lens too. If we are going in the direction of having to defend ourselves from quantum computing breaking our cryptography, we should do so without forgetting that attackers will keep using traditional techniques as well. A quantum-resilient cryptographic algorithm having trivial mathematical pitfalls is of no use to secure communications, as much as an algorithm which gets compromised the moment it is placed on any device.

A further aspect worth considering is that of performance. Post-quantum algorithms should offer at least comparable efficiency (e.g. in terms of time, memory) than the algorithms they substitute in order to be considered for widespread adoption. Imagine having to wait seconds or even minutes for every TLS-secure web page to be loaded by an Internet browser! Therefore, a careful performance analysis of newly introduced schemes is in order.

1.1.3 Where my research sits

The mix between the above two disciplines offers some interesting questions when examined jointly.

1. To what extent existing attacks in the literature of side-channel analysis apply to post-quantum algorithms, and how are the latter affected?

2. Are there existing countermeasures working in the context of post-quantum cryptography? Are there new possibilities?

Furthermore, a third interesting question stems from the necessity to bring post-quantum algorithms to real-world deployment.

3. How can post-quantum schemes be implemented efficiently?

Inspired by these research questions, I formulated two motivations which have been the main drivers of my contribution to the field throughout my years as a PhD student.

1. Post-quantum schemes, if they were to be deployed in place of classical ones, should offer comparable performance. This means that they should be implemented, assessed and optimised on as many platforms as possible to acquire a complete view over their potential applicability in the real world. Indeed, depending on the target platform, different trade-offs might be needed and some schemes might turn out to be more suitable than others.

2. Once they are implemented on any platform, post-quantum schemes are subject to the same implementation-specific attacks that threaten classical schemes. These include side-channel attacks, which try to recover information about a secret being used from time taken, power consumed and so on. An important evaluation aspect when deploying post-quantum schemes in the field, then, is to estimate which attacks they might be targeted by and how to defend them against effective attacks. Finally, such defensive mechanisms introduce an overhead that needs careful evaluation to establish their practicality.

I now explore in more detail how I contributed toward the resolution of the above questions, by diving into how my thesis is structured.

1.2 Structure of the thesis

Working in cryptography has been great for many reasons. What I have personally found to be the most intriguing one, however, is its being highly interdisciplinary. Contributions from many different branches of science and engineering are needed to make even the smallest statement about whether an algorithm is secure or not. In Chapter 2, I give an overview of such fields and the language that is used to combine them all: mathematics (Section 2.1). Thanks to it, I can talk about foundations of cryptography in Section 2.2, with a special focus on the theoretical constructions which I later analyse from a security point of view. I then specialise the discussion even more, by giving more details on the two branches at the intersection of which my thesis sits, post-quantum cryptography (with an emphasis on so-called *lattice-based systems*) and side-channel analysis, dealt with Sections 2.3 and 2.4 respectively.

I use the tools and definitions stated in Chapter 2 to build my contributions. Inspired by my second motivation, I start by addressing Question 1. In Chapter 3, I implement *Frodo*, which is the specific lattice-based post-quantum algorithm that forms the focus of this thesis, on the (emulated) *ARM Cortex-M0 embedded device*, together with the cor-

responding power consumption. I use two well known techniques from the side-channel arsenal to attack it. In Section 3.3 I apply a so-called divide-and-conquer attack, which targets each portion of a secret key individually and independently (hence the name). However, given the inherent dependency of certain internal computations on multiple portions of the key at the same time, I conclude that this is not the best approach an adversary would choose to attack the scheme under analysis, Frodo. This is why I adopt a more sophisticated approach called extend-and-prune in Section 3.4: the idea is that an adversary would start retrieving portions of secret key sequentially, from first to last, in order to leverage previously recovered secret chunks against later ones. The attack turns out to be devastating, which sheds some light on a deeper connection between what a side-channel adversary exploits versus what a traditional and mathematical adversary exploits. I highlight such a comparison in Section 3.5, which triggers a valuable discussion on parameter selection for this kind of algorithm.

Chapter 4 partially answers Question 2: once attacks have been covered, it is useful to look at the other side of the coin and explore ways to defend post-quantum algorithms. Interestingly, it turns out the that the extend-and-prune attack from Section 3.4 is as devastating as it is easy to mitigate: Section 4.1 contains a deterministic technique to break the underlying pattern exploited by it. I then take a different approach and devise a countermeasure that is particular to the matrices used by Frodo and by other post-quantum schemes, since some of them take values whose Hamming weight is low. I explore this direction in Section 4.2, where I describe a countermeasure to mask the Hamming weight of values, which therefore applies to attack scenarios where adversaries exploit the Hamming weight of secret elements. Coming up with new countermeasures, however, is not the only option to protect post-quantum schemes, as I show in Section 4.3 where I integrate known countermeasures from the literature into the the post-quantum algorithms under analysis. Finally, Section 4.4 compares all above countermeasures and lists pros and cons of each one.

Finally, it is now time to describe how I have contributed toward Question 3. Chapter 5 reports implementations (c.f. Section 5.3) and optimisations (c.f. Section 5.4) of several algorithms used in Frodo. On one hand, I carefully implement common subroutines using very fast low-level instructions, on the other hand I propose a modification to the original specifications of Frodo in such a way that a severe bottleneck, the generation of a large matrix, is overcome. For this chapter, I target the ARM Cortex-M4, which is considered to be small enough for many practical applications in the Internet of Things (IoT) ecosystem, yet powerful enough to run standard cryptographic protocols. On top

of the unprotected scheme, I also implemented all countermeasures from Chapter 4 in order to compare them from the perspective of their performance, see Section 5.5.

Every story must come to an end. I collect overall conclusions of my work in Chapter 6. On top of that, I make some historical consideration on the algorithms that are used in Frodo (which got modified when my work was being finalised) and I show how many of the conclusions I draw also apply to other algorithms in the literature involving lattice-based schemes.

1.3 List of papers

During the course of my PhD, I have released several works of mine. A list of papers whose content can be found in this thesis follows, while a list of papers of mine which have been excluded from this thesis can be found in Appendix B.

1. Together with Joppe Bos, Simon Friedberger, Elisabeth Oswald and Martijn Stam, I published the paper "Assessing the Feasibility of Single Trace Power Analysis of Frodo" at the Selected Areas in Cryptography (SAC) conference in 2018 [BFM$^+$18a]. In this paper, of which I am the main writer and contributor, my co-authors and I explored the aforementioned side-channel techniques, described in Chapter 3, to attack matrix multiplications by secret matrices. We applied such techniques to one scheme in particular, Frodo [BCD$^+$16, NAB$^+$17], which makes use of one public matrix whose values are uniformly random integers modulo a power of two, multiplied by a secret matrix whose entries are drawn from a very narrow distribution over the same space. This means that the possible entries are very limited compared to the size of the space, which is a favourable setting for side-channel adversaries. The content of this paper spans the whole of Chapter 3 and Section 4.1.

2. Together with the same co-authors, I released a paper in which we implemented Frodo on the embedded device called ARM Cortex-M4 [BFM$^+$18b]. Due to the nature of the algorithms involved, which I will detail in Chapter 2, a straightforward implementation is not possible due to the large dimensions of the public matrix mentioned above. We therefore optimised the subroutines in such a way that all algorithms fit on the device and exploit dedicated instructions. I was the main contributor in terms of implementation, experiments and writing. This is the paper that Chapter 5 covers. Note that, at the time of writing, an improved version of

the paper is undergoing writing. The version I am the main writer and content curator of is indexed as version 20181120 : 031936 on ePrint, while the new version is not included in this thesis because my contribution was marginal.

Chapter 2

Preliminaries

I would like to start the main body of this thesis by asking the reader an imagination effort. Metaphorically speaking, think of the book in your hands (or the file on your computer) as a house. Any house sits on the side of a street, which defines its geographical location and, to some extent, tells something about the dwellers. Inside the house there are rooms serving different purposes, be it cooking, sleeping or showering.

Chapter 1 specified the address, that is to say it placed this house into context. Along the lines of the metaphor, this house lies at a junction: it is located at the intersection of "lattice-based boulevard" and "side-channel lane", both in the "crypto neighbourhood". If you are familiar with theses like this one, it should come as no surprise how the coming chapters will be different rooms, each somewhat self-contained and dedicated to a specific task, all together making living (i.e. reading) this house hopefully pleasant. However, how does the current chapter fit into the metaphor?

Latin comes to the rescue, as the word "preliminaries" finds its roots in the combination of the preposition "prae-" and the noun "limen". The former simply means "before", while the latter, among other meanings, stands for doorstep. What is included in the preliminaries should then come before the doorstep, it should prepare visitors (i.e. readers) to what awaits them in the house (i.e. monograph). In other words this is the doormat of my thesis, whose job is to make your shoes and the concepts you will need as clear as possible.

With the above in mind, I will start by giving some basic notions of cryptography (Section 2.2), in order then to hint how they fall apart once quantum computers are included in the picture. Several reasons not to despair are promptly described for the faint-hearted (Section 2.4), among which the so-called lattice-based cryptography is scrutinised in more details. I will then glide towards a different area having little to

do with post-quantum cryptography per se but representing the second founding pillar of this thesis: side-channel analysis (Section 2.3).

2.1 Notation and Mathematical Background

Scalars are denoted by lower case italic letters, and are usually assumed to be in decimal representation. When the binary representation of an integer $i \in \mathbb{Z}$ is needed, it is denoted by $\langle i \rangle$. Vectors are denoted by lower case boldface letters, and matrices by upper case boldface letters. I adopt index notation to denote values in vectors and matrices, e.g. v_i is the ith component of vector $\mathbf{v}$ (and is not boldface because it is a scalar), while $M_{i,j}$ denotes the scalar in row i and column j of matrix $\mathbf{M}$. Note that the same holds for the binary representation of integers, in such a way that the lth bit of integer i is $\langle i \rangle_l$. To indicate the whole ith row of matrix $\mathbf{M}$, I will write $\mathbf{M}_i$; column j, instead, is denoted by $\mathbf{M}_j^\intercal$, i.e. the jth row of the transpose. I will make use of three special matrices: the identity matrix of size n, denoted by $\mathbf{I}_n$; the matrix filled with ones of dimension $n \times m$, denoted by $\mathbf{1}_{n \times m}$; the vector filled with n zeros, denoted by $\mathbf{0}_n$. Whenever vectors or matrices need to be parametric, I use superscript. For instance, if matrix $\mathbf{M}$ is parametrised by the parameter a, I write $\mathbf{M}^a$. Concatenation is denoted by $\|$.

I make use of two numerical sets only: the integers, $\mathbb{Z}$, and the reals, $\mathbb{R}$. For a strictly positive integer q, I denote by $\mathbb{Z}_q$ the remainders modulo q. All these sets are always assumed to have the classical operations of addition and multiplication, hence they are primarily used as groups, rings or (finite) fields.

Whenever a distribution $\mathscr{D}$ over a set S is used, I denote the operation of drawing a sample by $x \xleftarrow{\$} \mathscr{D}(S)$. In the case where entries of a matrix $\mathbf{M}$ of dimension $n \times m$ are drawn from the same distribution $\mathscr{D}(S)$, I write $\mathbf{M} \xleftarrow{\$} \mathscr{D}(S)^{n \times m}$. I denote by $\mathrm{Supp}(\mathscr{D}(S)) \subseteq S$ the support of the distribution, i.e. the subset of the domain of $\mathscr{D}$ whose elements have non-zero probability of being drawn. Note that when the set S is clear from the context, I drop it from the notation. I denote the probability of obtaining the specific outcome $x \in \mathrm{Supp}(\mathscr{D}(S))$ by $\Pr[X = x] = \mathscr{D}(x)$, where X is a random variable following $\mathscr{D}(S)$ and is often implicit from the context.

A particularly important distribution is the Gaussian distribution. I denote it by $\mathscr{N}(\mu, \sigma^2)$ if it has mean μ and variance σ^2, where σ is the standard deviation. The multivariate Gaussian distribution of dimension n, instead, is denoted by $\mathscr{N}_n(\boldsymbol{\mu}, \Sigma)$, with vector of mean $\boldsymbol{\mu}$ and covariance matrix Σ. Its probability density function is given

by

$$\Phi_{(\boldsymbol{\mu},\Sigma)}(\mathbf{x}) = \frac{1}{\sqrt{\det(2\pi\Sigma)}} \exp\left(-\frac{1}{2}\cdot(\mathbf{x}-\boldsymbol{\mu})\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\right) .$$

I will use several operations, the notation of which might be overloaded for the sake of maintaining bearable formalism. For instance, multiplication is either omitted or denoted by $\cdot$, which sometimes also indicates inner product between vectors or matrix multiplication. To avoid any confusion, the context will undoubtedly clarify which one is meant on a case-by-case basis.

2.2 Basic Cryptographic Notions

Nowadays, cryptography allows secure communication and interaction in many possible ways. The oldest of them all, dating back to ancient Greece and the Roman empire, is that of concealing the content of a message from malicious eyes. The classic example sees the character Alice wanting to exchange a message with Bob over an insecure channel, e.g. a battle field, the road between two *poleis*[1] or, more recently, the internet. The presence of a malicious actor, usually named Eve, is what makes the channel insecure. In such a basic example, the goal of Eve is to learn the content of the message just by eavesdropping on the channel. Such an adversary is called *passive*, and is the weakest of the possible threats cryptographic schemes are supposed to withstand.

It has always been understood that the only way of achieving such a simple form of confidentiality is by somehow transforming the original message, called *plaintext*, into something unintelligible, the *ciphertext*. Furthermore, the intended recipient needs a way of unravelling the ciphertext and learn the original message. Both the forward and backward transformations must be based on some form of secret piece of information only the legitimate parties possess, which is what prevents Eve from accessing the content of the communication. In modern terminology, the secret is usually named *key*.

Many notions in this chapter are covered from a high-level point of view, because many details are beyond the scope of this thesis. The interested reader is referred to Katz and Lindell's book [KL07], which extensively covers all topics mentioned here.

2.2.1 Symmetric-Key Cryptography

The first paradigm that arose historically can be formulated as follows. Under the initial assumption that Alice and Bob know the same secret, i.e. own the same key, then

[1] Plural of *polis*, the name modern historiographers give to an ancient Greek city-state.

Alice can use it to transform plaintext into ciphertext, performing an operation which is referred to as *encryption*, and Bob can reverse it performing *decryption*. On an intuitive level, it can be easily seen how this paradigm works by drawing an analogy with physical keys: if both Alice and Bob have a copy of the same key, then Alice can enclose a message in a box using a locker, which can be opened by Bob only.

This paradigm is called *symmetric-key encryption*, because the key used is symmetric, that is to say the same for both sender and receiver. It is particularly handy because secure channels are expensive (think of hiring an insured and guaranteed private shipping company), while insecure ones are cheap (e.g. using public postal service). Symmetric-key encryption allows a potentially very long message to be sent through a cheap and insecure method, while maintaining its confidentiality thanks to encryption. Only the key has to be sent by secure means, which is less of a problem being usually fixed and small(er) in size.

Slightly more formally, a symmetric-key encryption scheme is formed of three algorithms: key generation, encryption and decryption.

- KEYGEN(params) takes some publicly known parameters, and generates a symmetric key k;
- ENC(m,k) takes a plaintext m and a symmetric key k, and returns a ciphertext c;
- DEC(c,k) takes a ciphertext c and a symmetric key k, and returns a plaintext m'.

Correctness of the scheme requires $m = m'$. Note that this is a very simplified definition of a symmetric-key scheme, but amply suffices for the purpose of this thesis. A more precise definition, as well as what security means in the symmetric context, can be found in Katz and Lindell's book [KL07, Section 3.2].

Figure 2.1 graphically represents the three outlined algorithms and their use relative to each other. It is assumed that key generation is performed by the sender, who then sends the key over a secure channel (thick arrow) and an encrypted message over an insecure one (dashed arrow). The receiver simply applies decryption to retrieve the original message.

2.2.1.1 One-Time Pad

Arguably the most important example of symmetric encryption scheme is the *one-time pad*. First invented by Miller in 1882 [Mil82], the one-time pad derives its importance from the work of Shannon [Sha49], who proved its perfect secrecy. The latter property

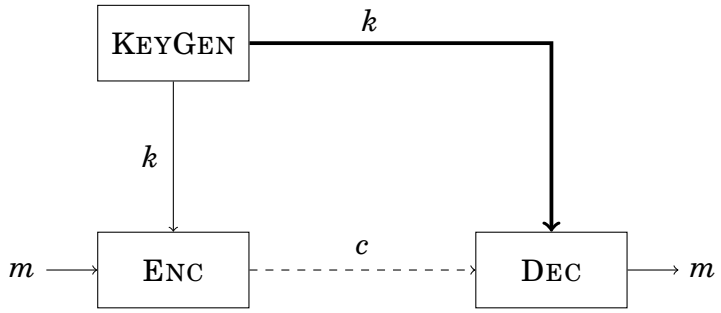


Figure 2.1: Schematic of the three algorithms forming a symmetric encryption scheme. Thick arrow represents a secure channel, while the dashed one stands for an insecure channel.

roughly signifies that absolutely no information about the plaintext is contained in the ciphertext.

A one-time pad is defined over fixed-length bitstrings, that is to say that the set from which keys, plaintexts and ciphertexts all come from is $\mathbb{Z}_2^n$ for a fixed positive integer n. The three algorithms forming any symmetric encryption scheme are instantiated as follows.

- KeyGen(n) picks a uniformly random key $k \xleftarrow{\$} \mathcal{U}(\mathbb{Z}_2)^n$, where the only parameter of the scheme is the length n;
- Enc(m, k) computes the ciphertext as the XOR between a message and the key, i.e. $c = m \oplus k$;
- Dec(c, k) simply reverts the above operation by computing the XOR between ciphertext and key, so that

$$c \oplus k = (m \oplus k) \oplus k = m \oplus (k \oplus k) = m \ .$$

Intuitively speaking, perfect secrecy follows from the fact that if an adversary sees a ciphertext and does not know which key was used, then any plaintext is possible under some key. As an example, consider $n = 8$, i.e. keys, plaintexts and ciphertexts are numbers between 0 and 255. If an adversary sees the ciphertext $c = 100$, she will never know whether the plaintext was $m = 7$ encrypted using $k = 99$, or if $m = 50$ encrypted using $k = 86$ because $7 \oplus 99 = 50 \oplus 86 = 100$. Since the key is equally likely drawn uniformly at random, any plaintext m could have produced the ciphertext c if the key $k = m \oplus c$ was used.

The cost to pay for such a strong security guarantee is prohibitively high, though. First of all, as the name suggests, each key should be used only once, for otherwise the XOR of two (or more) plaintexts is revealed. Furthermore keys should be truly uniformly random and should be as long as plaintexts. For these reasons, symmetric-key cryptography has developed in the direction of finding weaker yet realistically sufficient security definitions achieved by schemes more usable than the one-time pad.

In general the shared knowledge of k can be leveraged to achieve very effective and efficient algorithms in practice. However it is now time to discuss the clear problem behind this approach: how can Alice and Bob agree on a key? Alternatively, whoever runs KEYGEN needs a way to communicate the key to the other party. As briefly mentioned above, the expensive solution adopted in the past was to exchange it over a secure channel. Most of the times, this meant meeting in person or through dedicated and protected lines of communication. However limitations are evident, especially in the age of the Internet when machines located around the world are constantly in the need for secure communication.

2.2.2 Public-Key Cryptography

A major breakthrough in the field was made possible by the work of Diffie and Hellman [DH76] in 1976, who published for the first time a method allowing two parties to agree on a common secret over an insecure channel without previous communication: *public-key cryptography* (PKC) was officially born. It must be noted, however, that despite Diffie and Hellman were indeed the first authors to publish the idea of PKC, it was disclosed only in 1999 that three cryptographers at GCHQ, Clifford Cocks, James Ellis and Malcolm Williamson, had invented several methods (very similar to what now is known as Diffie-Hellman key agreement and RSA) to achieve PKC [gch]. Only the secrecy around their work prevented them from being recognised timely.

The core and ingenious idea behind the so-called Diffie-Hellman key agreement is that keys no longer need to be symmetric: two different but mathematically related keys are used, only one of which has to be kept secret (the *secret key*) while the other one is instead made public (the *public key*), hence the name public-key cryptography. In its original formulation, the Diffie-Hellman scheme did not allow arbitrary chosen messages to be sent, but was rather a method for two parties to agree on a common secret with the idea of using it as (a base for) a symmetric key.

Soon after the Diffie-Hellman protocol, in 1978, Rivest, Shamir and Adleman [RSA78] published an algorithm to send arbitrary messages without any previously agreed upon

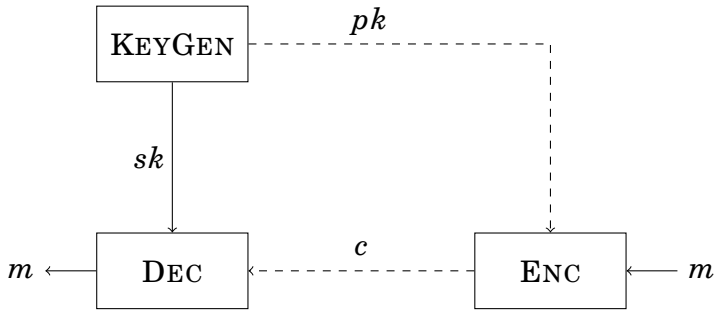


Figure 2.2: Schematic of the three algorithms forming a public-key encryption scheme. Dashed arrows represent information being sent over an insecure channel.

secret. RSA, named after the authors' initials, was the first public-key encryption scheme. Along the same line, ElGamal [ElG85] later modified the original Diffie-Hellman protocol to allow a similar functionality.

Similarly to symmetric-key schemes, public-key encryption is formed of three algorithms: key generation, encryption and decryption.

- KeyGen(params) takes some publicly known parameters and generates two keys, a public key pk and a secret key sk;
- Enc(m, pk) takes a plaintext m and a public key pk, and returns a ciphertext c;
- Dec(c, sk) takes a ciphertext c and a secret key sk, and returns a plaintext m'.

A more formal and comprehensive definition can be found in Katz and Lindell's book [KL07, Section 10.2].

Figure 2.2 represents the algorithms above and the relations among them. Compared to Figure 2.1, it should be visually immediate that no secure channel is needed: every communication happens over a potentially insecure channel.

Nomenclature Note. Terminology in the literature may vary. Sometimes what is here referred to as "public-key cryptography" is instead called "asymmetric-key cryptography", to stress that keys are different in contrast to symmetric-key cryptography. Similarly, the latter may be called "private-key cryptography". To avoid any confusion, I will exclusively use the terms that I have introduced here. Moreover in this text, public and secret keys always refer to those used in a public-key algorithms, while symmetric keys are used in the context of symmetric schemes.

2.2.2.1 The ElGamal Encryption Scheme

Classical public-key schemes are based on the hardness of certain mathematical problems from number theory. In their seminal work, Diffie and Hellman [DH76] built a mechanism to agree on a random number through exclusive exchange of messages over an insecure channel, with the idea that it could then be used in a symmetric encryption scenario. Their scheme is based on the supposed difficulty of a number theoretic problem, defined next.

Definition 2.1 (DLP)**.** Let G be a cyclic and multiplicative group of order q and g be its generator. For a given $h \in G$ the *Discrete Logarithm Problem* (DLP) asks to find the positive integer x such that $h = g^x$. The number x is called *discrete logarithm* of h.

As mentioned before, the discrete logarithm problem and the protocol Diffie and Hellman [DH76] built on it effectively opened the way to the rich body of research public-key cryptography has become. The next major milestone was achieved soon after, when the RSA cryptoscheme allowed encryption of arbitrary messages [RSA78], based on the hardness of factoring certain numbers. In 1984, ElGamal [ElG85] achieved a similar functionality to DLP. This is a particularly important scheme, as its structure is still used today in many modern cryptosystems. With reference to Figure 2.2, it works as follows.

- KeyGen(G, q, g) uses a cyclic and multiplicative group G of order q and generator g as parameters. The secret key is drawn as $x \xleftarrow{\$} \mathcal{U}(\mathbb{Z}_q)$, while the public key is computed as $h = g^x$;

- Enc(m, h) takes a plaintext $m \in G$ and the public key h, draws a uniformly random $y \xleftarrow{\$} \mathcal{U}(\mathbb{Z}_q)$ and computes two values:

$$c_1 = g^y$$
$$c_2 = h^y \cdot m \ .$$

  The ciphertext is simply the couple $c = (c_1, c_2)$;

- Dec(c, x) parses the ciphertext into its two components and retrieves the message as

$$m' = \frac{c_2}{c_1^x} \ .$$

Correctness immediately follows from

$$\begin{aligned} m' &= \frac{c_2}{c_1^x} \pmod{p} = \frac{h^y \cdot m}{(g^y)^x} \\ &= \frac{(g^x)^y}{(g^y)^x} \cdot m = m \end{aligned}$$

and from properties of exponentiation.

Despite having been designed in the early stages of public-key cryptography, the ElGamal encryption scheme is still an important and widely adopted cipher, most noticeably as part of the PGP protocol [Gar95]. Clearly, every practical implementation of the protocol needs to instantiate a group G where DLP is hard.

What makes ElGamal very relevant to this thesis, however, is its structure. At a closer inspection, the encryption algorithm uses a mechanism to hide the plaintext which should be familiar: h^y is effectively a somewhat random element in G being multiplied with the plaintext. This is very close to what the one-time pad does, but on the multiplicative group $(G, \cdot)$ rather than the additive group $(\mathbb{Z}_2^n, \oplus)$.

A crucial distinction with the one-time pad resides in how the "random" element is chosen: it is a truly uniformly random key k in the one-time pad, while it is a somehow constructed element h^y in ElGamal. This is a fundamental difference in cryptography, as the latter element is called *pseudorandom*, informally meaning that it is reasonably random-looking to a computationally bounded adversary. On the contrary, an information theoretic adversary, i.e. one with unbounded computational power, could simply brute-force all possible exponents of y until the value c_1 is found, and then use it to learn m from c_2.

The notion of pseudorandomness is tightly coupled with the difficulty of the underlying problem, instead of being unconditional (as is the notion of pure randomness). In particular, h^y is pseudorandom when given c_1, hence being reasonably good against computationally limited adversaries, only under the assumption that the so-called Decisional Diffie-Hellman (DDH) assumption holds. The latter is a computational hardness assumption very much similar to DLP (cf. Definition 2.1) but tailored to the more specific problem Diffie and Hellman [DH76] studied. Under the same premises of Definition 2.1, it asks to distinguish between (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) for uniformly random and independently chosen $a, b, c \in \mathbb{Z}_q$.

The above discussion is intentionally intuitive, because the formal details are beyond the scope of this thesis. The interested reader is referred to Katz and Lindell's book [KL07, Section 3.1], which describes at length all the proofs and definitions to truly

appreciate perfect secrecy, and exhaustively motivates the practical reasons why adversaries are usually limited in their computational power and why pseudorandomness is enough. Finally, be aware that what is described as "security" above only embraces a tiny fraction of the security notions available in modern cryptography. In fact, the above scheme is to be considered plainly insecure for any real-world purposes.

2.2.3 The KEM+DEM Framework

Adopting RSA or ElGamal encryption schemes to exchange messages might at first seem a valid alternative to pre-agreeing on a key and then using some symmetric-key scheme to encrypt a plaintext and decrypt a ciphertext: the former part is just cut off while the latter is achieved directly. In practice, however, public-key algorithms perform poorly compared to symmetric-key ones, especially when it comes to large messages. What has been established as the best method for communicating over insecure channels is then to use public-key cryptography to share a symmetric key, which then enables the parties to run much faster symmetric encryption and decryption algorithms. Such a division of roles takes the name *hybrid encryption paradigm*.

The naïve and plug-and-play solution just described, however, is problematic in practice: the potentially very different structures of public and symmetric cryptosystems might require several additional layers to render the above method functional. For example encrypting a, usually small, symmetric key might result in a ciphertext which is easy to break, thus requiring a padding function. This expands the attackable surface adversaries have at their disposal, sometimes to the point where the underlying strong mathematical security is bypassed and instantiations of a secure scheme are broken in practice [Ble98].

A better alternative is represented by the combination of a *Key Encapsulation Mechanism* (KEM) and a *Data Encapsulation Mechanism* (DEM). The overall idea is the same as before: the former is a public-key encryption scheme used to send a symmetric key, which is then used in the latter, being it a symmetric encryption scheme. Some crucial differences, however, exist.

- The symmetric key sent using the KEM is not interpreted as a message chosen by one party and to be encrypted, but is a random plaintext from the plaintext space of the chosen scheme.
- Once decrypted, the KEM ciphertext is fed into a *key derivation function* (KDF) to meet format requirements of the chosen DEM.

The KEM+DEM framework was first introduced as a hybrid paradigm by Cramer and Shoup [CS98], who used it to show a practical instantiation of their newly defined public-key encryption scheme. In its modern sense, thoroughly described and analysed by Shoup [Sho01], the functionality of a KEM+DEM framework is summarised by the following seven algorithms: a key generation, a key derivation function, encapsulation and decapsulation algorithms for the KEM part, and two similar ones for the DEM part.

- KEYGEN(params) takes some publicly known parameters and generates a public key pk and a secret key sk. This is essentially a key generation algorithm of a public-key scheme;

- KEM.ENC(pk) takes as input the public key and returns a shared key s and its encapsulation, denote by c_s;

- KEM.DEC(c_s, sk) takes an encapsulated key and the related secret key, returning the shared key s;

- KDF(s) takes a shared key and returns a symmetric key k;

- DEM.ENC(m, k) takes a plaintext m and a symmetric key k, and returns a ciphertext c;

- DEM.DEC(c, k) takes a ciphertext c and a symmetric key k, and returns a plaintext m';

As before, I omitted many formal details, which can be found in the seminal work by Cramer and Shoup[CS98] and in the later ISO standard proposal by Shoup [Sho01].

Figure 2.3 gives the last schematic of this chapter, somewhat combining those in Figures 2.1 and 2.2. Algorithms above the KDF form a public-key encapsulation scheme, while everything below is a symmetric one. The KDF itself makes sure that the shared secret s is transformed into a usable key under the symmetric scheme chosen to instantiate the DEM part of the framework. This ensures that the KEM encapsulation algorithm can pick a uniformly random "message", and not one chosen by the user which might turn out to be weak. On top of that, the benefits of both worlds are clear from Figure 2.3: no secure channels (thick arrows, cf. Figure 2.1) are needed, and messages are exchanged using the faster DEM, as opposed to encrypt messages using public-key encryption (cf. Figure 2.2).

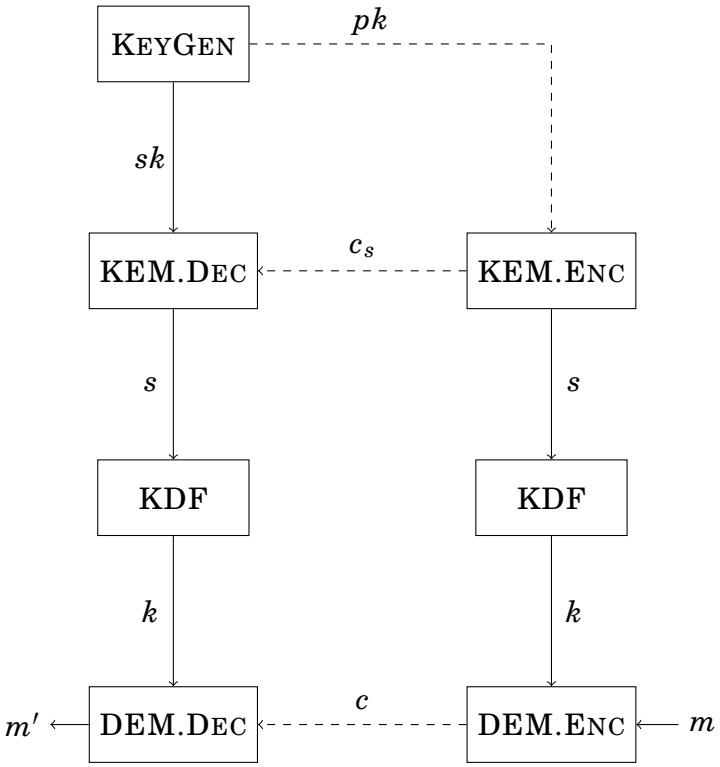


Figure 2.3: Schematic of the algorithms forming a KEM+DEM framework. Dashed arrows represent information being sent over an insecure channel.

2.3 Side-Channel Analysis

Historically, security notions have been built in such a way that a scheme, if proven secure, would withstand attacks against adversaries whose powers and view over the scheme were part of the notion itself. In other words, what an adversary is able to learn about the scheme, whether it is plaintext, ciphertext, signature and so on, was dictated by the security notions. It comes without saying that a proof of security against a certain adversary offers no guarantees about attacks from adversaries who have powers or behave differently than those for which the proof holds.

Kocher et al. [Koc96, KJJ99], in their seminal work, presented a class of adversaries that are more powerful than those captured by usual security notions, for which classical security proofs do not hold. Adversaries modelled by security proofs were only allowed to observe or tamper with the input and output of cryptosystems, as part of the so-called black box model. However, Kocher et al. [Koc96, KJJ99] showed that, whenever a scheme is implemented in the real world, adversaries can learn much more information by observing or tampering with the physical environment around the scheme.

Generally speaking, a *side channel* is any source of information from which adversaries can derive insights on the targeted scheme in order to learn secret material. Famous examples include time taken by an algorithm, power consumed by a device or electromagnetic radiations emanated from it. In this thesis, I only focus on power consumption, being it the most relevant for constrained devices that have been my main target throughout all my works. The information itself an adversary can learn from such channels is called *leakage*.

2.3.1 How devices consume power

An accurate and precise engineering description of how electronic devices work and how internal components and their physical characteristic affect security are out of scope for this thesis. I refer the interested reader to the comprehensive book by Mangard, Oswald and Popp [MOP07], which is considered to be among the best references for an introductory explanation of side-channel analysis and that also drives the discussion contained in the present section.

The power consumption of a device while performing one run of an algorithm is collected in a power trace, which is simply a vector of voltage values sampled during the time of execution. How many samples are collected per second, and therefore how long a power trace is, depends on the acquisition setup. Mangard, Oswald and Popp [MOP07, Section 4.1] model each point of a power trace as

$$P_{total} = P_{op} + P_{data} + P_{el.noise} + P_{const} \ .$$

A description of each component follows.

- P_{op} is the contribution to the overall power consumption that depends on the operation being performed. Clearly, different operations require different underlying logic to perform the designed function, therefore consume power differently.
- P_{data} is the contribution coming from the data being computed upon. Numbers are represented and stored differently depending on their value.
- $P_{el.noise}$ is the consumption of "everything else" on the device. When measuring multiple traces of the same operations over the same data, the fact that they still look different is due to this component. This component is also called environmental noise throughout this thesis.
- P_{const} is the least interesting component, and is inherent to the device itself.

Clearly $P_{el.noise}$ and P_{const} are effectively noise, i.e. they "corrupt" the power traces, and they are independent of any information an adversary wishes to learn. P_{op} and P_{data}, instead, are the components of the overall power consumption where useful information can be derived from. The extent to which such information is available and exploitable, however, is very difficult to state in general. Different attacks exploit different aspects of those two components, therefore a different model is needed. For a given attack scenario, the extent to which P_{op} and P_{data} are exploited is described by the following model

$$P_{op} + P_{data} = P_{exp} + P_{sw.noise}$$

where

- P_{exp} is the part which is exploitable by adversaries and that contains the information they target.
- $P_{sw.noise}$, called switching noise, is the part of the power consumption that is due to switching activity of those parts of the circuit which are independent of the exploitable data.

P_{exp} and $P_{sw.noise}$ can be a mix of P_{op} and P_{data}, and their actual nature depends on the attack scenario. These quantities allow the definition of a very useful metric, the *signal-to-noise ratio*, which helps quantify how much interesting information there is compared to the noise corruption:

$$\mathrm{SNR} = \frac{\mathrm{Var}[P_{exp}]}{\mathrm{Var}[P_{sw.noise} + P_{el.noise}]} \,.$$

Remark 2.1. P_{exp} includes all exploitable information that an adversary has (theoretically) available. However, as I will show later, the actual exploitation of all of P_{exp} requires careful characterisation of a device, which may not be practical. Thus in many attacks, adversaries only exploit parts of P_{exp}, which implies that "the other parts", which are statistically dependent and therefore cannot be included in $P_{sw.noise}$ either, remain unaccounted for. I detail the effects of this in Chapter 3, and refer to this phenomenon as *algorithmic variance*.

2.3.2 On power models

The previous subsection dealt with how to model a point in a power trace. These distinctions are useful also to understand how an adversary carries out an attack. Essentially all side-channel attacks follow the high-level description below.

- Power traces from the targeted device, which contains and uses a secret key, are collected. Depending on the attack scenario, these can be as few as only a single power trace up to several hundred thousands, with known or even chosen plaintexts.

- The adversary chooses a power model, that is to say a mathematical description of P_{exp}, based on targeted operations and data. Such a power model is then applied to all possible key, or subkey, values and to the known or chosen plaintexts.

- The corret key, or subkey, value is chosen according to some statistical tool that compares the power model and the real power traces.

The power model an adversary chooses plays a fundamental role in the description above, as it can make the difference between a devastating attack and a complete failure. The easiest option is to simply use an analytical function that is thought to approximate well enough the behaviour of the device. Mangard, Oswald and Popp [MOP07, Section 3.3] themselves mention a few of them, namely Hamming weight and Hamming distance of some values being computed upon. The intuition behind the effectiveness of these functions as power models is that certain internal components consume power depending on the number of bits that they handle. For instance, the power consumption of the bus is directly proportional to the bits it carries, thus the Hamming weight of numbers being sent, while registers switch, hence consume power, depending on which consecutive values are stored in them, which is proportional to their Hamming distance.

Unfortunately for adversaries, it is not always easy nor possible to approximate the consumption of every internal part of the device with said two functions. Therefore, a general procedure to derive a meaningful power model for the attack strategy and the target device is needed.

2.3.3 Computing the power model: templates

If the ready-to-go power models do not work for the operations and device under attack, the most general and comprehensive solution is to derive the power model from the device itself. This is known as profiling the device and is an operation which is rather expensive to perform and fairly heavy in assumptions. The idea of building a profile rather than using an existing one was put forward together with an attack exploiting such profiles.

Template attacks were first introduced by Chari et al. [CRR03] (although the term was already present in the paper by Kocher et al. [KJJ99]). The idea is that an adversary creates statistical descriptions, called templates, of the device's leakage for specific intermediate values by profiling the target device (or an equivalent one). Subsequently, one can use Bayesian methods (e.g. maximum likelihood estimation) to determine which template best matches the observed leakage, eventually leading to key recovery. In this sense, templates are used as power models in the description of the previous subsection.

I will give a more technical description of the mathematics behind templates in Section 3.2, where I will specialise the discussion on the algorithm I analyse and describe templates more in detail in that context.

2.3.4 Attack strategies

Once a power model is chosen based on the algorithm, the device and the operations under attack, an adversary applies it to key guesses and compares the result with real power traces. Statistical functions called distinguishers are used to then establish which of the key guesses is more likely to be correct, i.e. the one actually in use by the device. Different attacks adopt different distinguishers in order to succeed. However, there exist also multiple ways in which a distinguisher is applied. In the remaining of this thesis, I will make use of two of them in particular. The curious reader can learn more from the usual reference [MOP07, Chapters 5, 6, 8, 10].

2.3.4.1 Divide-and-conquer

This is the most classical attack strategy, dating back to the original paper by Kocher et al. [KJJ99]. As the name says, the approach followed by these types of attack is to split the key into independent subkeys and to retrieve them separately. The computational effort involved is therefore equal to the effort to retrieve one subkey times the number of subkeys, assuming no differences between subkeys.

The clear advantage is the efficiency of such a methodology, as well as the fact that it can be parallelised. The major drawback is that, in attacking subkeys separately, there is the implicit assumption that subkeys are independent and are operated on independently too. Sometimes this assumption simply holds, as is the case for, e.g., AES, while sometimes it is just an approximation to simplify the overall attack. As I will show in Chapter 3, the type of leakage stemming from the scenarios I analyse

depends on several subkeys simultaneously, thus opening the way for a different attack strategy.

2.3.4.2 Extend-and-prune

There are situations in which subkeys are independent, as in general having dependant subkeys might be problematic for security, but some internal operations compute over multiple subkeys and, therefore, useful information leakage on multiple subkeys exist.

In Frodo, this is true because, when performing a row–column multiplication, the accumulator depends on all positions, therefore updating it leaks much more than just one position. Chari et al. [CRR03] encountered a similar scenario while presenting templates attack against the cryposystem RC4, and therefore developed the extend-and-prune strategy.

The idea is fairly simple: instead of attacking positions independently from one another, the attacker proceeds in order from first to last. Information derived from earlier positions can be leveraged to formulate more precise and comprehensive guesses as part of the power model, therefore exploiting more information leakage from the power traces.

In practice, the extend-and-prune algorithm works as follows in the case of Frodo, where secret positions s_1 to s_n are targeted. Imagine a k-ary tree of depth n where the nodes at level i in the tree correspond to a partial guess $s_1, \ldots, s_{i-1}$ for the secret; for a given node at level i, its k out-going edges are labelled by the k possible values that s_i can take. This way, each path from the root to one of the k^n possible leaves uniquely corresponds to one of the possible values that the secret vector $\mathbf{s}$ can take. A distinguisher can sequentially calculate a score for a vector $\mathbf{s}$ by traversing the tree from the root to the leaf representing $\mathbf{s}$ where, for each edge it encounters, it cumulatively updates the score of $\mathbf{s}$. I defer a more mathematical description of the procedure to Section 3.2.

2.3.5 Other side-channels

The discussion so far has been rather biases toward power consumption as the vector of side-channel information. This is due to the fact that power consumption is the most interesting leakage source in the context of embedded devices, since it is rather practical to measure it. It is also central in this thesis, because many of the techniques discussed here had been proposed to leverage power consumption, despite being more general

than that. It is fair to say, however, that power consumption is not the only medium through which an adversary can gain insights on the intermediates of an algorithm.

Probably the second most exploited side-channel is time. Timing attacks were among the first to be reported by Kocher [Koc96]: different operations quite naturally take a different amount of time to be executed, however if such a difference depends on secret material (e.g. whether a bit is 0 or 1 [BB03]) then timing can be used to derive it. This is particularly relevant in lattice-based cryptography, as sampling is a crucial operation in any scheme while being also rather hard to secure from a side-channel standpoint. This was the case for a cache attack against the BLISS signature algorithm [BHLY16], in which the discrete Gaussian sampler was targeted. For this reason, constant-time samplers have always been a fairly hot topic in lattice-based cryptography, as demonstrated by a rich literature [KRR+18, MW17, ZSS19, PRR19]. Finally, electromagnetic emanation has also been used in the past to attack cryptosystems, and this has not changed with lattice-based cryptography [EFGT17c].

singtr

2.4 From Pre- to Post-Quantum Cryptography

Modern cryptography is based on the so-called Kerckhoffs's principle: the security of a scheme must rely on the secrecy of the key only. In particular, secrecy of the design and specifics of the scheme should not be relied upon when discussing its security. The immediate advantage is that sharing the scheme itself, e.g. communicating how the scheme works and should be used, can be done over a cheap and insecure channel, public parameters and operands might need some extra care and shared over an authenticated channel, while the expensive and secure channel is deployed to transmit the relatively small key only.

The principle also provides a very natural way to reason about the security of a scheme and the impact of attacks against it. If the key is everything which should be kept secret, then all schemes admit a trivial attack: trying all possible keys! This approach is called *brute-force attack* and its cost can be mathematically expressed by the number of keys an adversary would need to try in the worst case. If n is the number of bits forming the key, then the worst case is represented by enumerating all possible n-bit bitstrings, i.e. 2^n. Every attack performing worse than this threshold is usually not worth exploring, because a better alternative trivially exists. Conversely, any attack which finds the correct key in less than 2^n operations is deemed successful.

The above metric gives a fair ground on which to reason about security and to compare schemes, but does not represent the actual practicality of breaking a scheme: how quickly can a person enumerate, say, 2^{10} keys? The advent of computers drastically changed the answer to this question: a key-length that might be infeasible to enumerate by a person (a 4-digit PIN roughly corresponds to a $2^{13.28}$ enumeration effort) is likely to be broken in a fraction of a second by a computer. Complicating matters, computers get faster and faster, and key-lengths which were thought to be reasonably hard to break in the past, no longer are with the latest generation of computers. The most emblematic example is that of the *Data Encryption Standard* (DES), which is a symmetric encryption scheme standardised in 1977. At the time it was believed that a 56-bit key was enough, which however was brute-forced in less than a day in 1999, only 22 years after. Since 2005, DES is no longer a standard and has been replaced by the *Advanced Encryption Standard* (AES) [MVM09].

Nowadays internet communication uses 128-bit keys for symmetric ciphers and 3072-bit keys for some public-key ciphers (e.g. RSA) or 256-bit keys for others (e.g. Elliptic Curves). However, a revolution of even bigger proportions (in terms of breaking cryptography) than the advent of digital computers is around the corner: quantum computers.

The idea behind quantum computers is to encode information in some property of subatomic particles, which are then used in a physics experiment that mimic the behaviour of a program on a classical computer. At the end of the process a measurement takes place, the information is decoded back and an output is read. Programming quantum computers is different from programming classical ones, because the building blocks one can use to build "programs" differ. Some of them have been used to achieve massive speed-ups with respect to the classical counterparts, opening the way for faster algorithms targeting certain problems.

One such algorithm was published by Shor in 1997 [Sho97], and is particularly devastating for cryptography because, if a quantum computer powerful enough existed, it would allow efficient solutions to the DLP to be found, hence breaking all instances of the Diffie-Hellman protocol, of the ElGamal encryption scheme, and of RSA. As I discussed in Chapter 1, quantum computers are currently far from offering the computational power and the stability to harm any practical instantiations of the above schemes. Nonetheless, the process towards quantum-resilient algorithms is long and candidates have to go through a huge deal of scrutiny before being deemed safe to use, hence it certainly does not hurt to start well before the first large scale quantum computer is

available. Note that, although symmetric schemes are affected by some quantum algorithms, it is generally understood that the consequences are far less severe than in the case of public-key cryptography. Given a plaintext-ciphertext couple and considering the key of a symmetric cipher as an "unknown input", Grover's algorithm [Gro96] allows indeed to find it using only a number of calls to the function proportional to the square root of the input space. But this means that doubling the key-length, e.g. passing from a 2^{128} to a 2^{256} keyspace is enough to still ensure 128 bits of security.

There are several mathematical problems and cryptographic schemes which are being analysed by the community. Probably the first example of a post-quantum scheme was introduced shortly after Diffie and Hellman published their seminal work. McEliece [McE78] based a public-key encryption scheme on the hardness of decoding random code, which is widely believed to be an intractable problem even for quantum computers. This line of research is named *code-based cryptography*. Only one year after, Merkle [Mer79] proposed in his PhD thesis the first signature scheme, i.e. a way of using public-key cryptography to mimic the behaviour of a real-world signature, based on hash functions. *Hash-based cryptography* was born. Almost a decade later a new post-quantum scheme was devised, when Matsumoto and Imai [MI88] suggested to base cryptography on problems related to multivariate polynomials, paving the way for *multivariate cryptography*. Finally, the youngest of all proposals is without a doubt *Supersingular Isogeny Diffie-Hellman* (SIDH), which defines a protocol similar to the Diffie-Hellman key exchange but based on the hardness of computing isogenies of certain types of elliptic curves. The interested reader is referred to the seminal work by Jao and De Feo [JF11] for more details.

A further class of problems against which quantum computers are believed to offer little advantage is that of lattice problems. Ajtai [Ajt96] initiated the study by showing how cryptographic primitives could be based on problems defined over lattices. Because lattices and problems defined over them are the foundations of the security of algorithms analysed throughout this thesis, a more comprehensive focus will be given in Section 2.4.2.

2.4.1 NIST standardisation effort

As I briefly mentioned in Chapter 1, quantum computers are not quite mature and developed yet to harm cryptography, nonetheless achieving forward secrecy from today is enough of a good reason not to wait for the first quantum computer to be around to start building the post-quantum world. The latter is also a lengthy and complex

process, especially so because post-quantum algorithms are supposed to take the place of all public-key cryptosystems: commercial products, open source protocols, even the Internet itself have to be modified so to guarantee secure communication. Swapping all those algorithms, i.e. radically changing all protocols and solutions based on them, is not an overnight exercise and requires a long process of scrutiny and standardisation. Whatever candidate exists to take the place of classical algorithms, it must be proposed, evaluated, tested, parametrised and implemented before enough confidence about its security is grown. Elliptic curve cryptography offers an emblematic example: proposed by Miller [Mil86] and Koblitz [Kob87] in the eighties, it was not until the early two thousands that it found wide application. If the same time-to-market scale was assumed for post-quantum algorithms too, the moment of wide adoption might coincide with some estimates for when full-scale quantum computers will be around.

For these reasons, the National Institute of Standard and Technology (NIST) has made a first attempt to trigger a community-wide search for post-quantum algorithms that could be used instead of quantum-broken ones. The NIST post-quantum standardisation effort [Nat16a] gathered proposals of algorithms whose security is based on mathematical problems for which quantum computers are believed to offer no significant speed-up over classical computers. The NIST standardisation effort has been very important for my work, because some of my contributions were made towards scrutinising a particular class of cryptosystems which were submitted to it.

The process was initiated at the beginning of 2016 with a technical report from NIST highlighting the need for post-quantum cryptography [Nat16b], followed by a call for proposals detailing the “rules of game” [Nat16a] at the end of the same year. Groups from all over the world had time until the end of 2017 to design and submit post-quantum schemes along all relevant information to assess their goodness: security, performance, even side-channel resistance were all criteria according to which schemes would be judged. The first round saw 64 candidates, divided in the two categories Key Encapsulation Mechanism and Digital Signature, based on many different mathematical problems [Nat19]. At the beginning of 2019, candidates passing to the second stage of the standardisation process were announced: by that point many schemes were either withdrawn or found insecure by means of several attacks. Finally, the second round finished in July 2020 [Nat20] and the remaining candidates, now down to 15 in total, were divided in two classes per each category: “finalists” and “alternate candidates”. The former were deemed the most successful in terms of the metrics outlined above, while the latter, although not as appealing, were kept to offer trade-offs, for examples, in terms

of security offered over performance or because they were based on different mathematical problems than the finalists. Such a portfolio diversification has been indeed an overall goal of the standardisation process due to the fact that many problems have not sustained the "test of time", being them relatively recent.

The impact that such a rapidly evolving process has had on my thesis has been relatively mild. The scheme on which I focused most of my attention, Frodo, has reached the status of alternate candidate among the Key Encapsulation Mechanism, therefore all results contained in this thesis can be considered very much actual and still of interest for a scheme that might be considered for standardisation in the future.

2.4.2 Lattice-based Cryptography

A lattice is a mathematical structure that can be intuitively thought of as a set of points which are scattered around the space according to a fixed pattern. The formal definition follows.

Definition 2.2 (Lattice)**.** Let n be a positive integer. The set $\Lambda \subseteq \mathbb{R}^n$ is called a *lattice* if it is a discrete additive subgroup of $\mathbb{R}^n$.

The space I will be solely focusing on in this thesis is $\mathbb{R}^n$, hence lattice points are simply n-tuples of real numbers. The discreteness property formalises the idea that it is formed of scattered points, i.e. it is always possible to draw an n-dimensional sphere around a point such that it does not contain any other point in the lattice. Finally, the pattern I mentioned lattice points follow is mathematically expressed by the notion of an additive subgroup: the zero vector is always a lattice point and adding (or subtracting) lattice points must result in a lattice point. This conveys how the pattern looks, together with the notion of rank.

Definition 2.3 (Rank)**.** Let n and k be positive integers. Let $\Lambda \subseteq \mathbb{R}^n$ be a lattice, which is said to have *rank* k, or to be k-dimensional, if $\dim(\text{span}(\Lambda)) = k$. The latter simply means that Λ contains at most k linearly independent vectors. Moreover, Λ is called *full rank* if $k = n$, alternatively $\text{span}(\Lambda) = \mathbb{R}^n$.

The rank of a lattice gives an intuition on how the lattice is shaped in its host space. In the same way there can be a one-dimensional line living in a two-dimensional plane, there can be a one-dimensional lattice: it simply looks like a dotted line made up of equally spaced dots. All lattices involved in the cryptosystems described by this work are full rank, hence from now on I will always assume $k = n$. A very important property

the above definitions fail to catch is how to compactly describe a lattice, that is to say how to reasonably talk about, communicate or store a lattice.

Definition 2.4 (Basis of a Lattice)**.** Let n and k be positive integers and let $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\} \subseteq \mathbb{R}^n$ be k linearly independent vectors. Then

$$\Lambda(\mathbf{b}_1, \ldots, \mathbf{b}_k) = \left\{ \sum_{i=1}^{k} z_i \mathbf{b}_i : z_1 \ldots, z_k \in \mathbb{Z} \right\}$$

is a lattice and the set B is called a *basis* of the lattice Λ. The opposite also holds: for any given lattice Λ, any set of k linearly independent lattice points form a basis for Λ.

A basis fully specifies the whole lattice, as any other point can be obtained as a linear combination of the basis vectors using integer coefficients. Note that the actual components of a lattice point need not to be integer, but can be any real numbers. Any given lattice has infinitely many bases, but not all of them are regarded in the same way in lattice-based cryptography.

Definition 2.5 (Minimum Distance)**.** Let n be a positive integer and let $\Lambda \subseteq \mathbb{R}^n$ be a lattice. Its *minimum distance* is the length of a shortest non-zero vector, and is denoted by

$$\lambda_1(\Lambda) = \min_{\mathbf{v} \in \Lambda \setminus \{\mathbf{0}\}} \|\mathbf{v}\| \ .$$

The reason why the minimum distance is the length of *a* shortest non-zero vector is that, in general, the latter is not unique. For instance, the lattice $\mathbb{Z}^2 \subseteq \mathbb{R}^2$ has four such vectors: $(1,0)$, $(-1,0)$, $(0,1)$, $(0,-1)$ all of which have length $\lambda_1(\mathbb{Z}^2) = 1$. Intuitively speaking, a basis containing short vectors is considered a good basis, while a random-looking basis is bad. The difference is crucial: a good basis is likely to contain, or makes it very easy to find, a short lattice vector. As I shall detail now, such a problem is considered to be very difficult even for quantum computers if one starts with a random basis.

2.4.2.1 Hard Problems over Lattices

The first problem I discuss is precisely that of finding a short vector in a given lattice.

Definition 2.6 (SVP and SVP_γ)**.** Let n be a positive integer and let $\Lambda \subseteq \mathbb{R}^n$ be a lattice. The *Shortest Vector Problem* (SVP) asks to find a vector of length $\lambda_1(\Lambda)$ given (a representation of) Λ. In the γ-*approximate* SVP, instead, the vector to find can be of length at most $\gamma \cdot \lambda_1(\Lambda)$ for a given $\gamma \geq 1$.

Clearly SVP_γ is the same as SVP for $\gamma = 1$, and becomes less and less demanding as γ increases. Both of them are search problems, that is to say an explicit lattice vector has to be output to solve the challenge. The main hardness guarantee of cryptosystems contained in this thesis, however, is based on a closely related *promise problem*.

Definition 2.7 (GapSVP_γ)**.** Let n be a positive integer, $\Lambda \subseteq \mathbb{R}^n$ be a lattice and $d > 0$ be a real number. The *γ-approximate decisional SVP* (GapSVP_γ) asks to decide whether $\lambda_1(\Lambda) < d$ or $\lambda_1(\Lambda) > \gamma \cdot d$ for a given $\gamma \geq 1$.

What makes GapSVP_γ different from a usual decisional problem, where a simple binary answer is required, is that there is a non-trivial gap (hence the name "Gap") in the input space of the problem: when $\lambda_1(\Lambda)$ lies in the interval $[d, \gamma \cdot d]$ for the given Λ and the specified d. For such instances the problem is undefined and either answer is acceptable, although they are *promised* (hence the name "promise problem") not to be given, effectively labelling them as uninteresting.

The final problem I consider is only somewhat related to finding short integer vectors in a lattice, which is one of the possible approaches to solve it, but instead asks to find a lattice vector being the closest to a given non-lattice one.

Definition 2.8 (BDD_α)**.** Let n be a positive integer, let $\Lambda \subseteq \mathbb{R}^n$ be a lattice and let $\alpha > 0$. When given a vector $\mathbf{v} \in \mathbb{R}^n$ lying at distance (inherited from $\mathbb{R}^n$) at most $\alpha \cdot \lambda_1(\Lambda)$ from Λ, the *α-Bounded Distance Decoding* (BDD_α) asks to recover the closest lattice vector to $\mathbf{v}$.

As I will describe later on in this chapter, the main mathematical tool used by all cryptosystems in this thesis is a particular instance of BDD_α.

2.4.2.2 q-ary Lattices

Random real lattices are infinite structures by definition. This is problematic when they are used to implement cryptography on modern computers. Moreover, the fact that the coordinates of vectors take real values make their representation on computers cumbersome. For these reasons, lattice-based cryptography almost exclusively relies on another class of lattices, which take the role finite fields have in classical cryptography.

Definition 2.9. Let n and q be positive integers and let $\Lambda \subseteq \mathbb{R}^n$ be a lattice. Then Λ is a *q-ary lattice* if it satisfies $q\mathbb{Z}^n \subseteq \Lambda \subseteq \mathbb{Z}^n$.

Lattices of the above form have integer coefficients, making them very easy to implement on digital computers. Furthermore, requiring them to contain $q\mathbb{Z}^n$ allows componentwise equivalence classes modulo q to be defined, because if $\mathbf{v} \in \Lambda$ then so is $\mathbf{v}+(k_1 q, \ldots, k_n q)$ for any $k_i \in \mathbb{Z}$. Therefore it is possible to choose

$$\mathbf{v} \pmod q = (v_1 \pmod q, \ldots, v_n \pmod q)$$

as representative of the equivalence class. The lattice Λ becomes finite (modulo q) with at most q^n points.

Arguably the most appealing feature of q-ary lattices is that some hard problems can be specialised over their structure, resulting in more designer-friendly problems which, while still being based on very strong mathematical guarantees, in a sense "abstract away" the connection with the underlying lattice.

2.4.3 The Learning With Errors Problem

Hardness of the vast majority of modern lattice-based cryptography is based on a single problem, or variants thereof. Introduced by Regev in 2005 [Reg05], the *Learning With Errors* (LWE) problem has grown in popularity ever since. It has proven to be an extremely versatile mathematical tool on which to base cryptography, to the point of even making some previously uninstantiated applications possible [Gen09].

Definition 2.10 (LWE Distribution)**.** Let n and q be positive integers, and let χ be a distribution over $\mathbb{Z}$. Let $\mathbf{s}$ be a vector in $\mathbb{Z}_q^n$. The *learning with errors* (LWE) *distribution* $\mathcal{A}_{n,q,\chi}$ is defined to be the distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ that draws a vector $\mathbf{a} \xleftarrow{\$} \mathcal{U}\left(\mathbb{Z}_q\right)^n$ and a scalar $e \xleftarrow{\$} \chi(\mathbb{Z})$ and outputs the pair $(\mathbf{a}, b = \mathbf{a}\cdot\mathbf{s} + e \pmod q)$ where $\mathbf{a}\cdot\mathbf{s}$ denotes the inner product between vectors.

The LWE distribution, over which the problem is defined, has seemingly little to do with lattices. This has been considered a very attractive feature from an implementation perspective, because cryposystems based on LWE usually have a fairly simple structure and only require basic operations while still being based on solid hardness guarantees. If this sounds familiar, it is because there is a q-ary lattice behind the scenes.

Definition 2.11 (LWE Problem)**.** Let n, m and q be positive integers. Let χ be a distribution over $\mathbb{Z}$ and let $\mathbf{s} \in \mathbb{Z}_q^n$ be a uniformly random element in $\mathbb{Z}_q^n$. The *learning with errors* (LWE) *problem* $\mathrm{LWE}_{n,m,q,\chi}$, in its search version, asks to recover the vector $\mathbf{s}$

when given access to m samples from $\mathscr{A}_{n,q,\chi}$. The decisional variant, instead, asks to distinguish m samples $\left\{(\mathbf{a}_i, b_i) \xleftarrow{\$} \mathscr{A}_{n,q,\chi}\right\}_{i \leq m}$ from m uniformly distributed samples, i.e. drawn from $\mathscr{U}\left(\mathbb{Z}_q^n \times \mathbb{Z}_q\right)$.

2.4.3.1 Connection with Lattices and Hardness

The bridge between LWE and lattices is actually easier to build than it seems. Let $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ be the matrix whose rows are m samples drawn from the LWE distribution. Then an m-dimensional q-ary lattice can be defined as

$$\Lambda_q(\mathbf{A}) = \left\{\mathbf{x} \in \mathbb{Z}^m : \exists \mathbf{y} \in \mathbb{Z}_q^n \text{ such that } \mathbf{x} = \mathbf{A}\mathbf{y} \pmod q\right\} .$$

The columns of $\mathbf{A}$ form a basis for $\Lambda_q(\mathbf{A})$, hence the vector $\mathbf{As}$ is a lattice point in $\Lambda_q(\mathbf{A})$. The LWE distribution returns m samples $b_1, \ldots, b_m$, which are compactly denoted by the vector $\mathbf{b} \in \mathbb{Z}^m$. Its components are perturbed by integers drawn from χ, all together also denoted by a vector $\mathbf{e} \in \mathbb{Z}^m$. This means that with high probability (based on the distribution χ) $\mathbf{b}$ will not lie in $\Lambda_q(\mathbf{A})$. A solver is therefore asked to solve a BDD_α instance on the lattice $\Lambda_q(\mathbf{A})$.

A caveat in the above description is that the parameter α in the BDD_α problem is still unspecified. Intuitively speaking, α ensures that the lattice vector to be found is very near to the given target vector, in such a way that both existence and uniqueness of the former are guaranteed. This is precisely what the distribution χ in Definitions 2.10 and 2.11 is meant to achieve: it is chosen to be "narrow", i.e. concentrated around zero modulo q, in an attempt to ensure that each component of $\mathbf{e}$ is small. Thanks to precise bounds on the support of χ, it is possible to choose the parameter α to meet the requirements of the BDD_α problem in Definition 2.8.

Yet another possibility to show how the LWE problem relies on the hardness of lattice problems is by considering the following q-ary lattice

$$\Lambda_q(\mathbf{A}, \mathbf{b}) = \left\{\mathbf{x} \in \mathbb{Z}^m : \exists \mathbf{y} \in \mathbb{Z}_q^{n+1} \text{ such that } \mathbf{x} = (\mathbf{A} \| \mathbf{b})\mathbf{y} \pmod q\right\}$$

where $(\mathbf{A}\|\mathbf{b})$ denotes the matrix whose first n columns are the columns of $\mathbf{A}$, and the $n+1$st column is the vector $\mathbf{b}$ from the LWE distribution. By choosing $\mathbf{y} = (\mathbf{s}, 0)^\intercal$, it holds that $\mathbf{x} = \mathbf{As} \in \Lambda_q(\mathbf{A}, \mathbf{b})$. The choice $\mathbf{y} = (0, \ldots, 0, 1)^\intercal$ instead yields $\mathbf{x} = \mathbf{b} \in \Lambda_q(\mathbf{A}, \mathbf{b})$. Therefore it must hold that $\mathbf{e} = \mathbf{b} - \mathbf{As} \in \Lambda_q(\mathbf{A}, \mathbf{b})$. Recall that $\mathbf{e}$ has entries from the "narrow" distribution χ, meaning that $\Lambda_q(\mathbf{A}, \mathbf{b})$ is a q-ary lattice with very short vectors. With high probability $\mathbf{e}$ is in fact the shortest, thanks to choosing χ to be "narrow". Thus, under this formulation, LWE is equivalent to solving SVP in $\Lambda_q(\mathbf{A}, \mathbf{b})$.

These two techniques only hint to the reason how LWE is connected to hard problems defined over lattices, despite being simply defined as solving a system of noisy linear equations. The seminal work by Regev [Reg05] gives a full formal proof for why LWE is indeed very hard, based on the hardness of GapSVP_γ. I refer to it for the proof as well as for more formal details on the hardness of LWE.

A further appealing feature of LWE, shared with many other lattice problems, is the so-called *average-case to worst-case reduction*. A brief preamble is needed. Recall the description of a symmetric encryption scheme, visualised in Figure 2.1. The key k is often required to be chosen uniformly from random over the appropriate space, say $\mathbb{Z}_2^n$. If any key is truly equally likely, in particular a perfectly random-looking one has the same chances of being chosen as the zero key, i.e. $k = (0, \ldots, 0)$. However, many ciphers are easier to break in the latter instance. It is therefore possible to intuitively describe two cases: the *worst case* (from an adversary's perspective) is when the key looks indeed random. In practice though the *average case* applies: the key is extracted and "bad" instances, like the zero key, and "good" ones, say the random-looking key, are equally likely. So while in practice average-case instances are used, security should desirably be based on worst-case hardness. This notion is achieved by LWE: if adversaries can solve an average-case LWE instance (used in practice), then they must be so powerful to also be able to solve worst-case lattice problems, which are provably very hard. Clearly such powerful adversaries are not believed to exist (even with quantum computers), hence strengthening the confidence in the security of LWE.

2.4.3.2 Error Distribution

The error distribution that was originally put forward by Regev [Reg05] for LWE is the Gaussian distribution. Intuitively, it ensures that only values close to the expected value are drawn with non-negligible probability. To keep the well-behaved formulation of solving noisy linear equations modulo q, however, the support of χ has to be $\mathbb{Z}_q$. Regev therefore discretised the Gaussian distribution. In general there is no unique way of generating a discrete variant of a continuous distribution, as it depends on the properties to be maintained [Cha15]. The discretisation adopted by Regev works as follows: a sample from the Gaussian distribution is drawn, multiplied by q and rounded to the closest integer modulo q. In formulae, for all $z \in \mathbb{Z}_q$

$$\chi(z) = \Pr\left[\frac{z - 1/2}{q} \leq X \leq \frac{z + 1/2}{q}\right]$$

where $X \sim \mathcal{N}\left(0,(\alpha q)^2\right)$. The standard deviation αq is chosen such that $\alpha q > 2\sqrt{n}$, n being the length of the secret vector $\mathbf{s}$, in the main proof of hardness.

Since the seminal work, other choices for the error distribution have been supported by proofs of security, e.g. uniform distribution from bounded interval [MP13] and binary distribution [BLP+13].

2.4.3.3 Variants of LWE

Versatility is one of the features LWE is praised for. At the root of the considerable amount of applications it is possible to base on LWE, there is the number of variants LWE has been defined in. First and foremost, Applebaum et al. [ACPS09] showed that hardness is preserved if the secret vector $\mathbf{s} \in \mathbb{Z}_q$ is chosen from χ rather than uniformly at random.

Definition 2.12 (Normal form)**.** Let n, m and q be positive integers. Let χ be a distribution over $\mathbb{Z}$. The *normal form* LWE problem, in its search version, asks to recover the vector $\mathbf{s} \xleftarrow{\$} \chi(\mathbb{Z}_q)^n$ when given access to m samples of the form $(\mathbf{a}, \mathbf{a}\cdot\mathbf{s}+e)$ for a uniformly random $\mathbf{a} \xleftarrow{\$} \mathcal{U}\left(\mathbb{Z}_q\right)^n$ and a small error $e \xleftarrow{\$} \chi$. The decisional variant, instead, asks to distinguish m of the above samples from m uniformly distributed ones, i.e. drawn from $\mathcal{U}\left(\mathbb{Z}_q^n \times \mathbb{Z}_q\right)$.

Normal form allows the definition of new schemes, including those on which this thesis focuses, at almost no cost in security with respect to standard LWE. Along the same lines, reusing the matrix $\mathbf{A}$ for multiple secrets offers little to no advantage to an adversary, hence the next variant which employs a secret *matrix* rather than vector.

Definition 2.13 (Matrix-LWE)**.** Let n, $\overline{n}$, m and q be positive integers. Let χ be a distribution over $\mathbb{Z}$. The *Matrix-LWE* problem (in normal form) asks to recover the matrix $\mathbf{S} \xleftarrow{\$} \chi(\mathbb{Z})^{n\times\overline{n}}$ when given access to $(\mathbf{A}, \mathbf{AS}+\mathbf{E})$ for a uniformly random $\mathbf{A} \xleftarrow{\$} \mathcal{U}\left(\mathbb{Z}_q\right)^{m\times n}$ and an error matrix $\mathbf{E} \xleftarrow{\$} \chi(\mathbb{Z})^{m\times\overline{n}}$.

Matrix-LWE essentially is formed of $\overline{n}$ LWE instances in normal form sharing $\mathbf{A}$, i.e. $(\mathbf{A}, \mathbf{AS}_1^\intercal + \mathbf{E}_1^\intercal), \ldots, (\mathbf{A}, \mathbf{AS}_{\overline{n}}^\intercal + \mathbf{E}_{\overline{n}}^\intercal)$.

2.4.3.4 Structured versions of LWE and schemes based on them

A somewhat different direction has been taken by Lyubashevsky et al. [LPR10], who devised a way to further improve the size of the matrices. The main bottleneck of LWE

is the matrix $\mathbf{A}$, which needs to be of considerable size for concrete instantiations. It is not only large to send, but also to compute with. The solution involves more structure on top of the usual $\mathbb{Z}_q$, which I only describe here in its most basic version.

Let n be a power of two and $f(x) = x^n + 1 \in \mathbb{Z}[x]$, which is irreducible over the rationals thanks to the choice of n. Then the ring $R_q = \mathbb{Z}_q[x]/(f)$ is defined to be the ring of polynomials modulo f and where q is picked such that $q = 1 \pmod{2n}$ (this allows faster computations). On these premises, the following problem can be defined.

Definition 2.14 (Ring-LWE)**.** Let n and q be positive integers as above. The *Ring-Learning With Errors* (Ring-LWE) problem, in its search version, asks to retrieve a polynomial with uniformly random coefficients $s \in R_q$ when given access to samples of the form $(a, a \cdot s + e) \in R_q \times R_q$, where $a \xleftarrow{\$} \mathcal{U}(R_q)$ and the error term comes from a "small" distribution $e \xleftarrow{\$} \chi(R_q)$. The decisional variant asks to distinguish samples as above from uniformly random samples from $R_q \times R_q$.

The structure of Definitions 2.11 and 2.14 should appear immediately closely related. Indeed very similar hardness considerations follow for Ring-LWE too, and are contained in the seminal work by Lyubashevsky et al. [LPR10]. There are two crucial points worth mentioning on Ring-LWE. First of all, the lattice problems it bases its hardness on are no longer defined over generic q-ary lattices but over *ideal* q-ary lattices. This extra layer of algebraic structures is induced by the polynomial nature of Ring-LWE.

On the other hand, Ring-LWE exploits such an extra structure to encode more information in a single sample. To draw a rough and informal comparison, one Ring-LWE sample $(a, a \cdot s + e) \in R_q \times R_q$ corresponds to n samples of the form $(\mathbf{A}, \mathbf{As} + \mathbf{e}) \in \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n$ in the LWE setting. The reason behind this is that, for the particular case of $f(x) = x^n + 1 \in \mathbb{Z}[x]$, the polynomial $a \in R_q$ can be represented as a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ as

$$\mathbf{A} = \begin{pmatrix} a_1 & -a_n & -a_{n-1} & \dots & -a_2 \\ a_2 & a_1 & -a_n & \dots & -a_3 \\ \vdots & \vdots & \vdots & & \vdots \\ a_n & a_{n-1} & a_{n-2} & \dots & a_1 \end{pmatrix}$$

where $a_i \in \mathbb{Z}_q$ are the coefficients of the polynomial a. On top of saving a factor of n on the matrix $\mathbf{A}$, the chosen parameters n and q also allow fast polynomial multiplication via FFT-like (Fast Fourier Transform) computations. This fact will be particularly relevant in Section 5.5.5 when comparing an implementation of a scheme based on LWE

against a scheme based on Ring-LWE, because the scheme NEWHOPE is based on this problem.

As always, such improved performances do not come for free. The main source of scepticism around Ring-LWE is that adding algebraic structure to the problem means that the underlying mathematical assumptions are no longer defined over generic lattices but over ideal lattices, a subclass thereof. This means that there could potentially be attacks on the latter instances that do not apply in the generic case, this making Ring-LWE potentially less secure. To bridge the gap between the two, yet another problem has been defined which is a generalisation of both LWE and Ring-LWE.

Definition 2.15 (Module-LWE)**.** Let n and q be positive integers as before and let d be a positive integer. The *Module-Learning With Errors* (Module-LWE) problem, in its search version, asks to retrieve a polynomial with uniformly random coefficients $\mathbf{s} \in R_q^d$ when given access to samples of the form $(\mathbf{a}, \mathbf{a} \cdot \mathbf{s} + e) \in R_q^d \times R_q$, where $\mathbf{a} \xleftarrow{\$} \mathcal{U}\left(R_q\right)^d$ and the error term comes from a "small" distribution $e \xleftarrow{\$} \chi(R_q)$. The decisional variant asks to distinguish samples as above from uniformly random samples from $R_q^d \times R_q$.

2.15, even in this simplistic form, is very similar to 2.14. Indeed, Module-LWE [AD17, LS15, BGV14], from a very high level, entails to using small matrices of polynomials rather than large matrices of scalars (as done in LWE) or single polynomials (as done in Ring-LWE). This grants Module-LWE much more solid basis from a security perspective while still allowing a degree of implementation optimisation by means of FFT-like operations. Again, this will be relevant in Section 5.5.5 as SABER and KYBER are based on this problem.

2.4.3.5 Samplers used in lattice-based cryptography

All previous definition required a "narrow" distribution over $\mathbb{Z}_q$, i.e. a distribution whose samples are small with high probability. In Section 2.4.3.2 I specified that a discrete Gaussian distribution is used to this end, since it is by far the most popular choice and because is the one being adopted (or closely approximated) by the schemes analysed in this thesis. Sampling from such a distribution is not an easy task, especially when the quality of the samples, in terms of adherence to the theoretical distribution, has an impact on security, as is the case for digital signature based on lattices. In this section I outline several methods in which this can be achieved. I will give references for each of them for the interested reader, as samplers are not a central theme in this thesis and are reporter here for completeness only.

Bernoulli. The Bernoulli distribution returns 1 with a certain probability p and 0 with probability $1-p$. As trivial as this statement might sound, such a fact is used to sample from the discrete Gaussian distribution by performing several acceptance/rejection tests of uniform random variables with different probabilities ("biases") to obtain samples which are closer and closer to the target distribution. This usage of Bernoulli trials was proposed and utilised in the BLISS digital signature by Ducas, Durmus, Lepoint and Lyubashevsky [DDLL13].

Ziggurat. The Ziggurat method was originally proposed to sample from continuous distributions [MT00] and was later adapted to discrete distributions in order to be used in lattice-based cryptography [BCG$^+$14]. The idea is to approximate the curve of a (discrete) Gaussian distribution by rectangles of fixed and pre-computed sizes and then, similarly to the above method, utilise them to reject or accept uniformly distributed samples. This method was demonstrated on Lyubashevsky's signature [Lyu12].

CDT. The cumulative distribution function of a random variable X for any point x in its domain is the probability of the event $X \leq x$. For discrete distributions, this function can be computed on the discrete values the random variable can take and the results can be stored in a table T, from which the name *Cumulative Distribution Table* (CDT). Again, a uniformly random outcome is then tested against the value inside the table and the smallest index i such that $T[i]$ is bigger than outcome is returned. This is perhaps the most relevant sampler in this thesis as Frodo uses it.

Knuth-Yao. All previous method essentially assumed that a uniformly random variable could be produced and its outcome was then accepted or rejected based on some statistic or method. The Knuth-Yao sampler [KY76], instead, requires a uniformly random bitstring and builds a binary tree that has the possible sampled values as leaves. The bitstring is then used to derive a left/right path along the tree until a leaf is eventually reached and the corresponding value is returned.

2.4.4 Frodo

The NIST post-quantum competition [CJL$^+$16] has ignited a considerable amount of work in the academic community in the direction of bringing hard lattice problems (as well as other post-quantum candidates, briefly discussed before) into real-world cryp-

tosystems. Designers were asked to evaluate the security of their schemes in order to propose concrete parameters and instantiations meeting certain security levels. Implementations were mandatory too.

Among all theoretical problems which are considered to be hard against quantum computers, lattice-based ones were by far the most represented: more than a third of the submissions base their security on lattices. The LWE problem and variants almost monopolised the scene, with over a third of the schemes being based on them. As I have already mentioned, the Ring-LWE problem offers very attractive features when it comes to implementing schemes in practice. The fact that matrices are represented as polynomials reduces keys size, as well as opening the way to computations based on a variant of the Fourier transform.

Such practical improvements come at a cost, however. First of all, security is no longer based on generic lattices, but on a more restrictive class of lattices called *ideal* lattices. Such an extra structure is induced by the polynomial representation of matrices, and it is still an open question whether this structure is exploitable by attackers or not. Potentially there could be cryptanalysis that weakens the security of Ring-LWE while not affecting standard LWE in any significant way. A second drawback stems from the practical implementation of the Fourier transform. Several attacks over the years have targeted it, highlighting how delicate a step it is.

For these reasons, I will focus on peculiar properties of LWE-based schemes. In particular, one character will play a fundamental role in this thesis: Frodo, a scheme whose security is based on the matrix version of LWE in normal form. Note that multiple versions of Frodo exist: when I talk about it generically or on aspects which hold for all variants I simply call it Frodo. I write FrodoKEM [NAB+17] to mean the Key Encapsulation Mechanism submitted to round 1 of the NIST post-quantum competition. The version of Frodo before the NIST competition is referenced under the name FrodoKEP [BCD+16], which I briefly describe in Section 2.4.4.8. Finally, during the completion of this thesis FrodoKEM was modified as part of the second round of the NIST competition. While this last version is not covered by the coming chapters, I describe what has changed in Section 2.4.4.9. Note that the algorithmic pseudocode descriptions in this chapter differ slightly in notation and phrasing, but are otherwise taken directly from the Frodo specification [NAB+17].

2.4.4.1 An Old Structure for a New Scheme

For the sake of argument, I will now outline an amazingly insecure public-key encryption scheme which acts as a bridge classical cryptosystems and the new, post-quantum scheme Frodo [NAB+17, BCD+16].

- KEYGEN$(n, q, \chi, \overline{n})$ generates a uniformly random matrix $\mathbf{A} \xleftarrow{\$} \mathcal{U}\left(\mathbb{Z}_q\right)^{n \times n}$ and a matrix $\mathbf{S} \xleftarrow{\$} \chi(\mathbb{Z})^{n \times \overline{n}}$. It outputs $pk = (\mathbf{A}, \mathbf{B} = \mathbf{AS} \pmod q)$ as a public key and $\mathbf{S}$ as a private key.

- ENC$(pk, \mathbf{M})$ uses the public key pk to encrypt the message $\mathbf{M} \in \mathbb{Z}_q^{\overline{m} \times \overline{n}}$ as follows. First a matrix $\mathbf{S}' \xleftarrow{\$} \chi(\mathbb{Z})^{\overline{m} \times n}$ is drawn, then the ciphertext c is a couple formed of

$$\begin{aligned} \mathbf{C} &= \mathbf{S}'\mathbf{A} \pmod q \\ \mathbf{D} &= \mathbf{S}'\mathbf{B} + \mathbf{M} \pmod q . \end{aligned}$$

- DEC(sk, c) parses the ciphertext c into its two components and returns

$$\mathbf{M}' = \mathbf{D} - \mathbf{CS} \pmod q .$$

Correctness is trivial to verify, and so is insecurity: the secret key $\mathbf{S}$ can be derived from the public key using simple algebraic operations. What is interesting about the above non-crypto scheme is its structure: the second half of the ciphertext tries to mask the message by adding it to a matrix which is a function of the public key and some random matrix generated by encryption; the first half is generated to allow decryption at the receiver end. In other words, this is an insecure cousin of the ElGamal public-key encryption scheme. However, it fails at generating a pseudorandom public key from the secret key: this means that computing the latter from the former is easy, and that the message is not "one-time padded" with something difficult to predict by an adversary. The solution to all these problems is easier than it seems, thanks to the learning with errors problem.

2.4.4.2 Security Vs. Correctness

Matrix multiplications involved in the outlined insecure scheme can be seen as "error-less" Matrix-LWE samples, so a very first attempt to fix it is by adding error matrices. The patch would look as follows.

- KEYGEN$(n, q, \chi, \overline{n})$ generates a uniformly random matrix $\mathbf{A} \xleftarrow{\$} \mathcal{U}\left(\mathbb{Z}_q\right)^{n \times n}$ and two matrices $\mathbf{S}, \mathbf{E} \xleftarrow{\$} \chi(\mathbb{Z})^{n \times \overline{n}}$. It outputs $pk = (\mathbf{A}, \mathbf{B} = \mathbf{AS} + \mathbf{E} \pmod q)$ as a public key and $\mathbf{S}$ as a private key.

- ENC$(pk, \mathbf{M})$ uses the public key pk to encrypt the message $\mathbf{M} \in \mathbb{Z}_q^{\overline{m} \times \overline{n}}$ as follows. First matrices $\mathbf{S}', \mathbf{E}' \xleftarrow{\$} \chi(\mathbb{Z})^{\overline{m} \times n}$ and $\mathbf{E}'' \xleftarrow{\$} \chi(\mathbb{Z})^{\overline{m} \times \overline{n}}$ are drawn, then the ciphertext c is a couple formed of

$$\begin{aligned} \mathbf{C} &= \mathbf{S}'\mathbf{A} + \mathbf{E}' \pmod q \\ \mathbf{D} &= \mathbf{S}'\mathbf{B} + \mathbf{E}'' + \mathbf{M} \pmod q . \end{aligned}$$

- DEC(sk, c) parses the ciphertext c into its two components and returns

$$\mathbf{M}' = \mathbf{D} - \mathbf{CS} \pmod q .$$

Compared to the protocol before, this attempt really just draws error matrices of the right dimensions and adds them to already present matrix multiplications. Nonetheless, such a conceptually simple patch has a drastic impact on the race between correctness and security. On the one hand, the former is lost as

$$\mathbf{D} - \mathbf{CS} = \mathbf{S}'\mathbf{E} + \mathbf{E}'' + \mathbf{M} - \mathbf{E}'\mathbf{S} \pmod q$$

that is to say, the term $\mathbf{S}'\mathbf{AS}$ disappears as before but many multiplications among elements drawn from χ, i.e. "small", still remain. Therefore $\mathbf{M}' \neq \mathbf{M}$ and full correctness cannot be achieved. On the other hand, deriving the secret key from the public key now is equivalent to solving an instance of the Matrix-LWE problem, and the message is hidden by the term $\mathbf{S}'\mathbf{B} + \mathbf{E}''$ which is pseudorandom under the Matrix-LWE assumption, in the same fashion the term h^y in ElGamal was pseudorandom under the assumption that DDH is hard.

Security now relies on the difficulty of a mathematical problem, which is widely believed to hold. At the current stage, however, the scheme is unusable because each entry of the matrix $\mathbf{M}$ is perturbed by the corresponding entry of the matrix $\mathbf{S}'\mathbf{E} + \mathbf{E}'' - \mathbf{E}'\mathbf{S}$, which has a very high chance of being non-zero. Unfortunately, there is no way of using the above scheme while still retaining deterministic correctness, hence *probabilistic* correctness is required. This means that there will always be a non-zero probability of incorrect decryption, and it will be the job of a fine tuning of parameters to make it as small as possible.

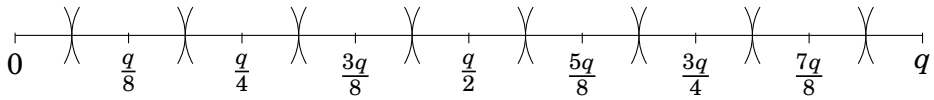


Figure 2.4: Threshold encoding/decoding for $B = 3$.

In order to withstand some noise, the message is encoded into $\mathbb{Z}_q$ rather than being chosen from it. For the sake of this description, I fix the modulus q to be a power of two, i.e. $q = 2^D$ for some positive integer D. This is consistent with the modulus adopted in FrodoKEM. The process starts with a message $\langle \mu \rangle \in \{0,1\}^B$, i.e. a bitstring that is of length a positive integer $B < D$. Alternatively, $\langle \mu \rangle$ is a number $0 \leq \mu < 2^B$, then *threshold encoding* is applied as

$$\begin{aligned} \mathrm{ec} : \left\{0, \ldots, 2^B - 1\right\} &\longrightarrow \mathbb{Z}_q \\ \mathrm{ec}(\mu) &\longmapsto q/2^B \cdot \mu \ . \end{aligned}$$

Intuitively, $\mathbb{Z}_q$ is partitioned into 2^B subsets, each containing 2^{D-B} elements. The arithmetic mean between the two borderline numbers of each set is the centre, and encodes a number between 0 and $2^B - 1$. Figure 2.4 illustrates how $\mathbb{Z}_q$, represented as a line, is split in the special case $B = 3$. By convention, a number sitting at the intersection of two sets belongs to the set starting from it. Note that the line really is an opened circle, since $q = 0 \pmod q$ the two extrema are the same point.

Vice versa, *threshold decoding* takes a number in $\mathbb{Z}_q$ and outputs the centre of the set it belongs to. This is mapped back to a number between 0 and $2^B - 1$ as

$$\begin{aligned} \mathrm{dc} : \mathbb{Z}_q &\longrightarrow \left\{0, \ldots, 2^B - 1\right\} \\ \mathrm{dc}(c) &\longmapsto \lfloor 2^B/q \cdot c \rceil \pmod{2^B} \end{aligned}$$

where $\lfloor x \rceil = \lceil x + 0.5 \rceil$ is the rounding function, taking a positive real number and returning the nearest integer (this formulation is consistent with the borderline numbers convention described before). If the message of the probabilistic encryption is encoded using this method, it is possible to allow for some errors to occur. Decryption will now succeed as long as the error is not too large, i.e. if the decrypted ciphertext still lies in the same set of the plaintext that was sent. In other words, the introduced error should not exceed $q/2^{B+1}$ in absolute value.

This is where the description of Frodo begins and where the insecure scheme outlined in Section 2.4.4.1 is turned into a fully-fledged post-quantum scheme. First of all, to be consistent with matrix dimensions in the scheme, the plaintext must be a matrix

Algorithm 1 FrodoKEM.Encode

Input: Bitstring $\boldsymbol{\mu} \in \{0,1\}^{B \cdot \overline{n} \cdot \overline{m}}$
Output: Matrix $\mathbf{M} \in \mathbb{Z}_q^{\overline{m} \times \overline{n}}$

1: **for** $i \leq \overline{m}$ **do**
2: **for** $j \leq \overline{n}$ **do**
3: $m \leftarrow \sum_{l=1}^{B} \mu_{((i-1)\overline{n}+j-1)B+l} \cdot 2^l$
4: $M_{i,j} \leftarrow \mathrm{ec}(m) = q/2^B \cdot m$
5: **return M**

Algorithm 2 FrodoKEM.Decode

Input: Matrix $\mathbf{M} \in \mathbb{Z}_q^{\overline{m} \times \overline{n}}$
Output: Bitstring $\boldsymbol{\mu} \in \{0,1\}^{B \cdot \overline{n} \cdot \overline{m}}$

1: **for** $i \leq \overline{m}$ **do**
2: **for** $j \leq \overline{n}$ **do**
3: $\mu \leftarrow \mathrm{dc}(M_{i,j}) = \lfloor 2^B/q \cdot M_{i,j} \rceil \pmod{2^B}$
4: **for** $l \leq B$ **do**
5: $\mu_{((i-1)\overline{n}+j-1)B+l} \leftarrow \langle m \rangle_l$
6: **return** $\boldsymbol{\mu}$

$\mathbf{M} \in \mathbb{Z}_q^{\overline{m} \times \overline{n}}$, meaning that the encoding scheme has to be applied entry-wise with the same criteria: every B bits are encoded into a $\mathbb{Z}_q$ element. In total, then, a message can be up to $B \cdot \overline{n} \cdot \overline{m}$ bits long. Each position $M_{i,j} \in \mathbb{Z}_q$ is perturbed by $\mathbf{S}'_i \mathbf{E}_j^\intercal + E''_{i,j} - \mathbf{E}'_i \mathbf{S}_j^\intercal \pmod{q}$, which has to be less than $q/2^{B+1}$ in absolute value for the correct plaintext to be decoded. Algorithms 1 and 2 show the final encoding and decoding processes, respectively.

2.4.4.3 Pseudorandomness Generation

Both uniformly random matrices, e.g. $\mathbf{A}$, and those from χ, e.g. $\mathbf{S}$ and $\mathbf{E}$, would require a prohibitive amount of true randomness in practice. The designers of FrodoKEM have solved this issue by using a *Pseudo Random Number Generator* (PRNG): a truly random, usually short, *seed* is given as input and an arbitrarily long pseudorandom output is generated. There are two main advantages: the amount of true random bits is limited to creating seeds, which are much shorter than full matrices, and communication overhead is decreased since only short seeds need to be exchanged while matrices are generated offline.

I shall start by describing how the uniform pseudorandom matrix $\mathbf{A}$ is generated,

Algorithm 3 FrodoKEM.Gen-AES128

Input: Seed $\text{seed}_{\mathbf{A}} \in \{0,1\}^{128}$
Output: Pseudorandom matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$

1: **for** $i \leq n$ **do**
2: **for** $j \leq n$, $j \leftarrow j+8$ **do**
3: $\mathbf{p} \leftarrow \langle i \rangle \| \langle j \rangle \| 0 \ldots 0 \in \{0,1\}^{128}$
4: $\langle c_{i,j} \rangle \| \ldots \| \langle c_{i,j+7} \rangle \leftarrow \text{AES}_{\text{seed}_{\mathbf{A}}}(\mathbf{p})$ where $\langle c_{i,k} \rangle \in \{0,1\}^{16}$
5: **for** $k \leq 8$ **do**
6: $A_{i,j+k-1} \leftarrow c_{i,j+k-1} \pmod q$
7: **return A**

Algorithm 4 FrodoKEM.Gen-cSHAKE128

Input: Seed $\text{seed}_{\mathbf{A}} \in \{0,1\}^{128}$
Output: Pseudorandom matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$

1: **for** $i \leq n$ **do**
2: $\langle c_{i,1} \rangle \| \ldots \| \langle c_{i,n} \rangle \leftarrow \text{cSHAKE}(\text{seed}_{\mathbf{A}}, 16n, 2^8 + i)$ where $\langle c_{i,j} \rangle \in \{0,1\}^{16}$
3: **for** $j \leq n$ **do**
4: $A_{i,j} \leftarrow c_{i,j} \pmod q$
5: **return A**

and later focus on matrices drawn from χ, as the process is quite different. In the implementation of FrodoKEM, when $\mathbf{A}$ is sampled as $\mathbf{A} \xleftarrow{\$} \mathcal{U}(\{0,1\})^{n \times n}$, what actually happens is that only a 128-bit seed $\text{seed}_{\mathbf{A}}$ is drawn uniformly from random and $\mathbf{A}$ is deterministically generated from it by a PRNG. The saving in true randomness is huge, considering that $\mathbf{A}$ effectively requires $D \cdot n^2$ bits, i.e. between 5.9 MB and 14.5 MB depending on the parameter set.

In the specifications [NAB+17], the algorithm which takes as input a $\text{seed}_{\mathbf{A}}$ and returns $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ is called FrodoKEM.Gen and comes in two flavours, depending on the PRNG that is used to expand the seed. Algorithm 3 uses AES [MVM09] and works as follows. Two indexes i, j are converted in binary and padded to reach a length of 128. The resulting bitstring is encrypted with AES using $\text{seed}_{\mathbf{A}}$ as a key. The resulting 128-bit long ciphertext is interpreted as sixteen 16-bit numbers modulo q and stored in $A_{i,j}, \ldots, A_{i,j+7}$. This is repeated for all rows, indexed by i, and every eight columns, indexed by j. Note that the j-loop runs over the columns of $\mathbf{A}$, therefore jumps eight positions at a time because that is the number of consecutive positions generated.

The second possibility is using cSHAKE [KjHCP16] and is described in Algorithm 4.

Algorithm 5 FrodoKEM.Sample

Input: A random 16-bit number r and the appropriate function T_χ
Output: A sample $e \in \mathbb{Z}$ from χ

1: $t \leftarrow \sum_{l=2}^{16} \langle r \rangle_l \cdot 2^l$
2: $e \leftarrow 0$
3: **for** $z \leq s$ **do**
4: **if** $t > T_\chi(z)$ **then**
5: $e \leftarrow e + 1$
6: $e \leftarrow (-1)^{\langle r \rangle_1} \cdot e$
7: **return** e

In this case cSHAKE is initialised with $\text{seed}_{\mathbf{A}}$ and with a customisation value which only depends on the row index, and produces n 16-bit numbers modulo q which are stored as one full row of $\mathbf{A}$. Note that to generate $\mathbf{A}$, cSHAKE is always used in its 128-bit variant. Note that the factor 2^8 is an arbitrary constant introduced in the specifications [NAB+17].

Algorithms 3 and 4 are very different and clearly not compatible with one another. A fair comparison in performance of the two algorithms is also quite hard to establish due to the large number of possible implementations of the two PRNGs. On the one hand, AES is exceptionally fast when the underlying platform has dedicated instructions to run it, e.g. AES-NI instructions on Intel platforms. On the other hand, when they are not available, cSHAKE seems to offer better performance [NAB+17].

2.4.4.4 Sampling from χ

Pseudorandom sequences, as output by AES and cSHAKE in Algorithms 3 and 4 respectively, can be directly used to generate matrices from the uniform distribution, but an extra step is needed to generate matrices from the distribution χ. First of all, cSHAKE128 or cSHAKE256 (depending on the chosen parameter set, AES is never used) are applied to expand a seed into a pseudorandom sequence of the correct length. This is split into a number of 16-bit pseudorandom numbers, which are fed into an inversion sampling algorithm to produce a sample from χ.

Algorithm 5 describes how inversion sampling transforms a random number r into a sample from χ by using its probability mass function, here represented by T_χ. The for loop, which is concretely implemented in constant time to avoid timing leakage, returns a value $e \in \{0, \ldots, s\}$. The last step uses the least significant bit of the random number r to multiply e by a sign.

Algorithm 6 FrodoKEM.SampleMatrix

Input: A seed $\text{seed}_{\mathbf{E}} \in \{0,1\}^S$, dimensions n_1, n_2, the appropriate function T_χ and a domain separator c
Output: A matrix sample $\mathbf{E} \in \mathbb{Z}^{n_1 \times n_2}$ from $\chi^{n_1 \times n_2}$

1: $\langle r_1 \rangle \| \ldots \| \langle r_{n_1 n_2} \rangle \leftarrow \text{cSHAKE}(\text{seed}_{\mathbf{E}}, 16 n_1 n_2, c)$
2: **for** $i \leq n_1$ **do**
3: **for** $j \leq n_2$ **do**
4: $E_{i,j} \leftarrow \text{FrodoKEM.Sample}(r_{(i-1)n_2+j}, T_\chi)$
5: **return** $\mathbf{E}$

Algorithm 7 FrodoKEM.KeyGen

Input: None.
Output: Key pair $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B}) \in \{0,1\}^{128} \times \mathbb{Z}_q^{n \times \overline{n}}$ and $sk = (\mathbf{s}, \mathbf{S}) \in \{0,1\}^{\text{Len}} \times \mathbb{Z}_q^{n \times \overline{n}}$.

1: $\mathbf{s} \| \text{seed}_{\mathbf{E}} \| \mathbf{z} \xleftarrow{\$} \mathcal{U}(\{0,1\})^{3\text{Len}}$
2: $\text{seed}_{\mathbf{A}} \leftarrow \text{cSHAKE}(\mathbf{z}, 128, 0)$
3: $\mathbf{A} \leftarrow \text{FrodoKEM.Gen}(\text{seed}_{\mathbf{A}})$
4: $\mathbf{S} \leftarrow \text{FrodoKEM.SampleMatrix}(\text{seed}_{\mathbf{E}}, n, \overline{n}, T_\chi, 1)$
5: $\mathbf{E} \leftarrow \text{FrodoKEM.SampleMatrix}(\text{seed}_{\mathbf{E}}, n, \overline{n}, T_\chi, 2)$
6: $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E} \pmod q$
7: **return** public key $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B})$ and secret key $sk = (\mathbf{s}, \mathbf{S})$

Algorithm 6 describes the final algorithm that, given a seed $\text{seed}_{\mathbf{E}}$, returns a matrix $\mathbf{E}$ distributed according to χ. The algorithm allows for matrices of any dimension to be generated, hence the extra inputs n_1, n_2, and can use the same seed to generate distinct matrices. The latter feature is made possible by the domain separator c, which completely changes the output pseudorandom sequence of cSHAKE even if the same seed is used. Multiple matrices can be then produced from the same truly random seed, hence further saving on the cost of true randomness. Finally, note that in this case cSHAKE is used either in its 128-bit or in its 256-bit variant depending on the parameter set. Since this is mostly irrelevant for this thesis, I avoid specifying the variant.

2.4.4.5 Frodo Algorithm Descriptions

It is finally time to explicitly state the three main algorithms forming FrodoKEM. Being a KEM, it is formed of key generation, encapsulation and decapsulation, cf. Figure 2.3.

Algorithm 7 is the key generation algorithm of FrodoKEM, which is very similar to that of the toy scheme presented in Section 2.4.4.2. There are two main differences: how

Algorithm 8 FrodoKEM.ENCAPS

Input: Public key $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B}) \in \{0,1\}^{128} \times \mathbb{Z}_q^{n \times \overline{n}}$.

Output: Ciphertext $c = (\mathbf{B}', \mathbf{C}, \mathbf{d}) \in \mathbb{Z}_q^{\overline{m} \times n} \times \mathbb{Z}_q^{\overline{m} \times \overline{n}} \times \{0,1\}^{\text{LEN}}$ and shared secret $\mathbf{ss} \in \{0,1\}^{\text{LEN}}$.

1: $\mu \xleftarrow{\$} \mathcal{U}(\{0,1\})^{\overline{m} \times \overline{n} \times B}$
2: $\text{seed}_{\mathbf{E}'} \| \mathbf{k} \| \mathbf{d} \leftarrow \text{cSHAKE}(pk \| \mu, 3\text{LEN}, 3)$
3: $\mathbf{S}' \leftarrow \text{FrodoKEM.SampleMatrix}(\text{seed}_{\mathbf{E}'}, \overline{m}, n, T_\chi, 4)$
4: $\mathbf{E}' \leftarrow \text{FrodoKEM.SampleMatrix}(\text{seed}_{\mathbf{E}'}, \overline{m}, n, T_\chi, 5)$
5: $\mathbf{A} \leftarrow \text{FrodoKEM.Gen}(\text{seed}_{\mathbf{A}})$
6: $\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}' \pmod q$
7: $\mathbf{E}'' \leftarrow \text{FrodoKEM.SampleMatrix}(\text{seed}_{\mathbf{E}'}, \overline{m}, \overline{n}, T_\chi, 6)$
8: $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}'' \pmod q$
9: $\mathbf{C} \leftarrow \mathbf{V} + \text{FrodoKEM.Encode}(\mu) \pmod q$
10: $\mathbf{ss} \leftarrow \text{cSHAKE}(\mathbf{B}' \| \mathbf{C} \| \mathbf{k} \| \mathbf{d}, \text{LEN}, 7)$
11: **return** Ciphertext $c = (\mathbf{B}', \mathbf{C}, \mathbf{d})$ and shared secret $\mathbf{ss}$

Algorithm 9 FrodoKEM.DECAPS

Input: Ciphertext $c = (\mathbf{B}', \mathbf{C}, \mathbf{d}) \in \mathbb{Z}_q^{\overline{m} \times n} \times \mathbb{Z}_q^{\overline{m} \times \overline{n}} \times \{0,1\}^{\overline{m} \times \overline{n} \times B}$, secret key $sk = (\mathbf{s}, \mathbf{S}) \in \{0,1\}^{\text{LEN}} \times \mathbb{Z}_q^{n \times \overline{n}}$ and public key $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B}) \in \{0,1\}^{128} \times \mathbb{Z}_q^{n \times \overline{n}}$.

Output: Shared secret $\mathbf{ss} \in \{0,1\}^{\text{LEN}}$.

1: $\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}'\mathbf{S} \pmod q$
2: $\mu' \leftarrow \text{FrodoKEM.Decode}(\mathbf{M})$
3: **return** FrodoKEM.CCA(c, $\mathbf{s}$, μ', pk)

matrices at (pseudo)random are generated and the fact that the secret key contains the vector $\mathbf{s}$ as well. This will be used in Algorithm 10 to make the scheme secure against chosen ciphertext adversaries [KL07, Section 10.6].

Algorithm 8 specifies the encapsulation algorithm. The value $\boldsymbol{\mu}$ is drawn uniformly at random and encoded using Algorithm 1. The latter turns it into a matrix, which is then hidden by $\mathbf{S}'\mathbf{B} + \mathbf{E}''$, which is pseudorandom under the assumption that Matrix-LWE is hard (and that cSHAKE is a secure PRNG). The sender then retains $\boldsymbol{\mu}$ and transmits the encapsulated key.

Decapsulation, in Algorithm 9, allows the uniform pseudorandom matrix $\mathbf{S}'\mathbf{A}\mathbf{S}$ to be cancelled out, then applies the decoding procedure described in Algorithm 2. As long as the introduced error does not exceed the critical threshold of $2^{B+1}/q$, the two parties will indeed share the bitstring $\mathbf{ss}$. This can then be fed into a key derivation function and the communication can happen securely thanks to the DEM part of the protocol.

Algorithm 10 FrodoKEM.CCA

Input: Ciphertext $c = (\mathbf{B}', \mathbf{C}, \mathbf{d}) \in \mathbb{Z}_q^{\overline{m} \times n} \times \mathbb{Z}_q^{\overline{m} \times \overline{n}} \times \{0,1\}^{\text{LEN}}$, secret vector $\mathbf{s} \in \{0,1\}^{\overline{m} \times \overline{n} \times B}$, recovered vector $\mu' \in \{0,1\}^{\overline{m} \times \overline{n} \times B}$ and public key $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B}) \in \{0,1\}^{128} \times \mathbb{Z}_q^{n \times \overline{n}}$.
Output: Shared secret $\mathbf{ss} \in \{0,1\}^{\text{LEN}}$.

$\text{seed}_{\mathbf{E}'} \| \mathbf{k}' \| \mathbf{d}' \leftarrow \text{cSHAKE}(pk \| \mu', 3\text{LEN}, 3)$
$\mathbf{S}' \leftarrow \text{FrodoKEM.SampleMatrix}(\text{seed}_{\mathbf{E}'}, \overline{m}, n, T_\chi, 4)$
$\mathbf{E}' \leftarrow \text{FrodoKEM.SampleMatrix}(\text{seed}_{\mathbf{E}'}, \overline{m}, n, T_\chi, 5)$
$\mathbf{A} \leftarrow \text{FrodoKEM.Gen}(\text{seed}_{\mathbf{A}})$
$\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}' \pmod q$
$\mathbf{E}'' \leftarrow \text{FrodoKEM.SampleMatrix}(\text{seed}_{\mathbf{E}'}, \overline{m}, \overline{n}, T_\chi, 6)$
$\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}'' \pmod q$
$\mathbf{C}' \leftarrow \mathbf{V} + \text{FrodoKEM.Encode}(\mu') \pmod q$
if $\mathbf{B}' = \mathbf{B}''$ and $\mathbf{C} = \mathbf{C}'$ and $\mathbf{d} = \mathbf{d}$ **then**
    return ss $\leftarrow \text{cSHAKE}(\mathbf{B}' \| \mathbf{C} \| \mathbf{k}' \| \mathbf{d}', \text{LEN}, 7)$
else
    return ss $\leftarrow \text{cSHAKE}(\mathbf{B}' \| \mathbf{C} \| \mathbf{s} \| \mathbf{d}', \text{LEN}, 7)$

	NIST1	NIST2
n	640	976
$q = 2^D$	2^{15}	2^{16}
$\overline{n} = \overline{m}$	8	8
B	2	3
LEN	128	192
Failure	$2^{-148.8}$	$2^{-199.6}$

Table 2.1: Parameter sets of FrodoKEM.

FrodoKEM is secure against chosen ciphertext adversaries [KL07, Section 10.6]. For convenience of representation, I included the extra steps required to achieve such a level of security separately in Algorithm 10.

2.4.4.6 Parameter sets

Algorithms 7 to 10 can be instantiated in practice with two parameter sets, which are summarised in Table 2.1. I call them NIST1 and NIST2, and they target the NIST-defined 128 (tier 1) and 192 (tier 3) security levels respectively. Integers n, $\overline{n}$ and $\overline{m}$ determine the matrices' dimensions, q is the modulus, B the number of exchanged bits encoded in each entry of the plaintext, while the last row indicates the probability of failure, i.e. the probability that there are any discrepancies between the $\boldsymbol{\mu}$ as generated

	Probability of (in multiples of 2^{-16})											
	0	±1	±2	±3	±4	±5	±6	±7	±8	±9	±10	±11
NIST1	9456	8857	7280	5249	3321	1844	898	384	144	47	13	3
NIST2	11278	10277	7774	4882	2545	1101	396	118	29	6	1	

Table 2.2: Probability mass function of χ for both parameter sets.

by the encapsulation and as derived from decapsulation.

As already mentioned, the failure probability is determined by the distribution χ. The importance of adopting a "narrow" distribution for secrets and errors should be clear from the discussion on correctness: the retrieved matrix $\mathbf{M}'$ is perturbed by the error matrix $\mathbf{E}''' = \mathbf{S}'\mathbf{E} + \mathbf{E}'' - \mathbf{E}'\mathbf{S}$, which is a function of the matrices drawn from χ. If any of those matrices were too large, then entries of $\mathbf{E}'''$ would exceed the bound $q/2^{B+1}$ required for decoding.

So the distribution χ should be chosen such that elements in $\mathbf{E}'''$ are within bound with very high probability. Moreover, the security of the whole scheme should provably be based on Matrix-LWE. The designer of FrodoKEM therefore chose one such distribution, which additionally allows efficient sampling: a discrete and symmetric distribution approximating a rounded continuous Gaussian. For a positive integer $s \in \mathbb{Z}$, the support of χ is $\mathrm{Supp}(\chi) = \{-s, \ldots, s\}$. Every integer z which lies inside the support is drawn from χ with probability listed in Table 2.2. Each row of the latter can be interpreted to be a function $T_\chi : \mathrm{Supp}(\chi) \to \mathbb{Z}$, one per parameter set: the probability of a value $z \in \mathrm{Supp}(\chi)$ is simply computed as $\chi(z) = T_\chi(z)/2^{16}$. Note from Table 2.2 that $s = 11$ for NIST1 and $s = 10$ for NIST2. Since $s \ll q$, χ is guaranteed to produce numbers in $\mathbb{Z}_q$ despite being formally defined over $\mathbb{Z}$.

2.4.4.7 Security

The interested reader is referred to the specifications of FrodoKEM [NAB+17] for details on the security aspects of FrodoKEM. Note that the distribution χ used here is slightly differs from the "narrow" distribution I defined for LWE in Section 2.4.3. In general such a discrepancy detail might be problematic, because it could potentially signify that the proof of security no longer holds for the parameters FrodoKEM adopts. In the specification [NAB+17], however, this aspect is taken care of and an alternative proof based on a variant of BDD_α is provided.

	CCS1	CCS2	CCS3	CCS4
n	352	592	752	864
$q = 2^D$	2^{11}	2^{12}	2^{15}	2^{15}
$\overline{n} = \overline{m}$	8	8	8	8
B	1	2	4	4
Failure	$2^{-41.8}$	$2^{-36.2}$	$2^{-38.9}$	$2^{-33.8}$

Table 2.3: Parameter sets of FrodoKEP before the NIST competition.

	Probabilities (times 2^x)						
	0	±1	±2	±3	±4	±5	±6
CCS1	88	61	20	3			
CCS2	1570	990	248	24	1		
CCS3	1206	919	406	104	15	1	
CCS4	19304	14700	6490	1659	245	21	1

Table 2.4: Probability mass function of χ before the NIST competition.

2.4.4.8 Frodo before NIST

Frodo made its first appearance in the literature well before the NIST competition took place. Bos et al. [BCD+16] originally proposed Frodo as a Key Exchange Protocol (KEP) based on the hardness of Matrix-LWE in normal form. The core operations are essentially the same as the NIST candidate, however different parameter sets were proposed.

Tables 2.3 and 2.4 contain the parameter sets that were put forward in the original publication [BCD+16]. Note that, despite mostly having the same names, these parameters are not interchangeable with those in Tables 2.1 and 2.2 because the protocol in the original publication is radically different. Comparing Tables 2.1 and 2.3, it can be noted how different the probabilities of failure are: this is only due to the security notions the two schemes meet. FrodoKEP and FrodoKEM [NAB+17] are hardly comparable as full schemes, however many internal operations were maintained. Since this thesis mainly focuses on such internal operations, there will be no ambiguity and arguments will go through for every version of Frodo. In fact, many conclusions will hold for generic matrix multiplications in LWE-like contexts, hence embracing more schemes than just Frodo, as I will explain in Chapter 6.

Despite being superseded, FrodoKEP is very important for this thesis. One reason is historical: I had started working on the ideas explored in Chapter 3 before the NIST competition began, which resulted in most of the content published in the corresponding

paper of mine to be focused on FrodoKEP. The other main reason is that FrodoKEP is such that there are no long-term secrets involved: every session has its own ephemeral secret matrices. This motivated the study for a single-trace setting, where only limited information can be gathered by an adversary. Such a distinction with FrodoKEM, where a long term secret does exist, is behind the evolution of many arguments from single-trace to a multi-trace setting, the latter being the typical scenario of DPA (see Section 2.3).

2.4.4.9 Frodo after this thesis

Much like FrodoKEM has a predecessor, FrodoKEP, it also has a successor. Once the main body of my research and experiments were over, and the writing of this thesis was well under way, the NIST post-quantum competition proceeded one step further and entered the second round. Frodo is among the schemes that survived the scrutiny, and was slightly updated to incorporate insights and feedback from the community [NAB+19]. In Chapter 6, I will detail to what extent my work applies to the updated scheme. At then of the second round, Frodo was also chosen as an alternate candidate in round 3 [Nat20].

2.5 Literature Review

Lattice-based cryptography is a relatively young field in the broader cryptography literature, as its roots can be dated back to the nineties. From an historical point of view, there were several works that pre-dated many of the constructions discussed in this thesis and must therefore be acknowledged and credited for initiating the body of work I will dive into. The first in line was the cryptosystem called NTRU, designed by Hoffstein, Pipher and Silverman around 1996 [HPS96] and later refined and published in 1998 [HPS98]. NTRU is formed of operations among polynomials, which is why it is usually listed among the "structured" lattice-based cryptosystems. Apart from being among the first schemes based on lattices, NTRU is particularly relevant today because, differently than other schemes proposed at the time, it is still considered a valid candidate, to the point that several submissions of the NIST post-quantum standardisation effort are based on it or variants thereof [ZCH+19, BCLv19]. Another, yet analogous, reason why NTRU is still very relevant nowadays is the considerable amount of works that analysed practical aspects of the scheme such as implementation performance [Sil99] and security [HS98, SW06a]. An honourable mention should also go to other cryptosystems

that have not aged as well but that inspired many subsequent works, including those most relevant to the topics of this thesis. These include the works by Ajtai [Ajt96], Ajtai and Dwork [AD97], and Goldreich, Goldwasser, Halevi [GGH97a, GGH97b].

The Learning With Errors problem was introduced in 2005 by Regev [Reg05], along with a public-key cryposystem based on it. What LWE added to the picture is a way of working with lattices under the efficient blanket of modular arithmetic: on the one hand the security guarantees come from solid mathematical theory, on the other all a machine needs to be instructed to do is (mostly) modular sums and multiplications. It should come as no surprise that, from then on, a race to bring lattice-based cryptography to the real world started.

Another milestone was hit by Lyubashevsky, Peikert and Regev [LPR10] who introduced the Ring-LWE problem. Very loosely speaking, in its simplest form (which is sometimes called Poly-LWE) Ring-LWE considers the fact that instead of using a public uniformly random matrix over $\mathbb{Z}_q$, it is possible to draw only a uniformly random vector and then build a circulant matrix from it. While this is only the intuition on the surface, what happens to the underlying lattice is that it is *ideal*. The precise meaning of this word is beyond the scope of this thesis, many details are given in the original paper [LPR10], but in essence ideal lattices are a special kind of lattice, therefore it might be the case that doing so weakens the security guarantee of schemes based on such a problem. So far, no ones has demonstrated a significant difference in solving LWE and Ring-LWE, which are commonly treated as equivalently secure for any practical purpose.

The fact that Ring-LWE uses vectors, actually polynomials, rather than matrices means that public and secret keys are shorter, which is very appealing when it comes to implementing a scheme. Also, a transformation exists, the Number Theoretic Transform (NTT), such that both addition and multiplication between polynomials can be performed component-wise. Again, this is very convenient from a performance perspective. It should not be surprising, therefore, that cryptosystems based on Ring-LWE were the first to find practical applications [dCRVV15, CMV+15]. Bos et al. [BCNS15] showed Ring-LWE could be used inside the TLS protocol, which eventually inspired the creation of NewHope [ADPS16], a Key Exchange Protocol which was later featured in an experiment by Google to deploy post-quantum cryptography to secure internet communications [Bra16].

Being fast and rather compact are two key features which also drove the implementation of lattice-based schemes, particularly those based on Ring-LWE, on more

constrained environments. For example, Alkim, Jakubeit and Schwabe [AJS16] implemented NewHope on the ARM Cortex-M0 and Cortex-M4, while Oder and Güneysu [OG17] implemented it on a FPGA. Clearly, implementations of schemes based on unstructured lattices, other than those described in this thesis, appeared as well, like that of Howe at al. [HOKG18]. I mentioned the latter paper in Chapter 5 too as being particularly relevant to compare my own implementation. In fact, the main difference between our works being that Howe et al. did not use SIMD instructions. This idea has also been deployed in the context of Ring-LWE by Kannwischer, Rijneveld and Schwabe [KRS19] who exploited them to speed up polynomial multiplications on the ARM Cortex-M4.

Implementations on smaller and smaller devices, however, call for side-channel analysis. A fairly wide range of them has already been successfully applied to lattice-based schemes at large, by exploiting execution time [SW06b], cache hits and misses [BHLY16, Pes16], power consumption [PPM17, ATT+18] and electromagnetic radiation [EFGT17b]. The work by Primas, Pessl and Mangard [PPM17] is particularly interesting as they mount a single-trace attack, much like in the same spirit as in Chapter 3, against the NTT of a typical Ring-LWE public-key scheme. Interestingly, this exploits a peculiarity of structured lattices, which in this case do offer more attack surface, even if only for side-channel adversaries. Also, they claim traditional masking would be ineffective against their attack. In the context of LWE, instead, Aysu at al. [ATT+18], as I already mentioned in Chapter 3, performed a similar attack to my extend-and-prune one: they targeted a FPGA implementation and retained all possible guesses for the first position of the secret, but then continued with only a single guess, which is what I called laser beam in Chapter 3, for the others. In this sense, I improved upon their work in the flexibility of the attack. Non-invasive side-channel attacks, like those mentioned so far, are not the only concern in the context of lattice-based cryptography. Fault attacks are invasive techniques which try to destabilise intermediate computations by tampering with the device (e.g. by injecting high voltages or lasers). They seem to find a fertile ground against several algorithms of lattice-based schemes [VPR19], being them signing [EFGT17a], decryption [KY11] or error sampling [BBK16]. The latter case is precisely what my co-authors and I have tried to defend against in our paper [HKM+19] to which, as I mentioned in Chapter 1, I contributed too little to be included in this thesis. In essence, we propose to apply several statistical tests to the output of the sampler, e.g. checking for repeating sequences (in order to thwart zeroing), computing sample mean and variance (to check whether they adhere to the theoretical secure values) or perform a chi-square test (which is more stringent on the actual distribution in output).

Any faults, programming errors or erroneous activity leading to an unusual distribution would then be caught.

As always in the field of cryptography, works followed to patch the situation: several countermeasures were also published. These range from blinding by Reparaz et al. [RdCR+16], who used the fact that randomness could homomorphically be applied to the ciphertext, to masking schemes [RRdC+16, OSPG18]. In this context, it is interesting to notice how the last two papers take very different approaches when it comes to shifting values back to the unmasked domain: Reparaz et al. [RRdC+16] developed a rather complex probabilistic decoder which applies non-linear threshold decoding and accounts for errors by repeating experiments to transform arithmetic shares of the secret to boolean shares of the plaintext, while Oder et al. [OSPG18] unmasked the ciphertext sequentially, therefore outputting directly the unmasked plaintext.

Chapter 3

Analysis of Single-Trace Power Attacks against Frodo

The conceptual path followed by Section 2.4.4 to introduce Frodo stresses how ideologically similar some post-quantum schemes are to classical, as in pre-quantum, cryptosystems. They essentially share the same structure built upon different mathematical problems. It should not be too surprising, therefore, that some attack vectors which are essentially problem-independent still apply and need to be analysed. This is the case of side-channel attacks which, as introduced in Section 2.3, apply whenever something sensitive is implemented, no matter where the security guarantees come from.

The current and next chapters describe the main core of the work I did during my PhD, which was precisely aimed at assessing to which extent existing techniques in the side-channel analysis domain apply, or do not apply, to post-quantum schemes.

The majority of the content of this chapter was published at the Selected Areas in Cryptography (SAC) 2018 conference [BFM+18a]. My coauthors were Joppe Bos, Simon Friedberger, Elisabeth Oswald and Martijn Stam. I was the main author of the paper and the main contributor to content, writing and experiments.

3.1 Overview

Some background is useful to understand why the research described in this chapter took the form presented here. The choice to focus on template attacks stemmed from the fact that it is not always possible to use usual unprofiled power models, e.g. Hamming weight and distance. This way I achieved a good level of generality. The extend-and-prune methodology was chosen because it is the technique that best fits the incremental

nature of the matrix multiplication carried out in LWE-based schemes. Indeed, it shows the best outcome from an adversarial point of of view.

It must be noticed, however, that the assumption about unprofiled power models not succeeding is purely determined by the setting, as there are examples where they work just fine. In the context of post-quantum cryptosystems, this is the case in the work by Aysu et al. [ATT+18], who demonstrated the efficacy of horizontal Correlation Power Analysis (CPA) in a single trace setting against Frodo's matrix multiplication $\mathbf{AS}$ when implemented in hardware. Their attack assumes knowledge of the architecture in order to target specific intermediate registers. They further assume that the Hamming distance is a good approximation of *their specific* device's leakage. They do rely on an extend-and-prune strategy, but they do not further explore challenges that may arise in contexts where the device's leakage is too far from Hamming weight/distance for an unprofiled method to work.

My coauthors and I filled this gap by investigating single-trace attacks against software implementations of "ring-less" LWE-based constructions, as used by Frodo. We focused our attention on matrix multiplications as they are fundamental in any plain-LWE scheme and because it is where secret matrices are used the most. Moreover, Frodo is designed with some side-channel considerations that limit the attack surface, thus making our target choice very relevant: algorithms run in constant-time, NTT transforms cannot be used, hence cannot be the target of side-channel attacks [PPM17], and the sampling of secret matrices, which is usually rather delicate, is done via a simple CDT sampler, that are based on tables and implementable in constant time, apart from being targeted already by other works in the literature [KH18]. All in all, matrix multiplications indeed are a very natural candidate for attackers to choose.

When Frodo is used as a key agreement protocol, the secret $\mathbf{S}$ is ephemeral and the calculation of $\mathbf{AS}+\mathbf{E}$ that I target is only performed once (or twice), resulting in only a single trace. This limited usage implies that only a subset of side-channel techniques apply. When Frodo is used as a KEM, the overall private key (of the KEM) *is* used repeatedly for decapsulation and the usual techniques relying on a variable number of traces do apply. However, even then my work provides useful guidance on security, and indeed, my results can be translated to any "small secret" LWE scheme, that is, any scheme where the individual entries of $\mathbf{S}$ are "small" in the space over which the scheme is defined. I will give more details about this aspect in Chapter 6.

Even if only a single trace corresponding to $\mathbf{AS}+\mathbf{E}$ is available, each *element* in $\mathbf{S}$ is still used multiple times in the calculation of $\mathbf{AS}$, enabling so called horizontal dif-

ferential power analysis. Here the single trace belonging to **AS** is cut up into smaller subtraces corresponding to the constituent $\mathbb{Z}_q$ operations. Hence, the number of subtraces available for each targeted $\mathbb{Z}_q$ element (of **S**) is determined by the dimension of the matrix **A**. For square **A** as given by the suggested parameters, this immediately leads to a situation where high dimensions for **A**, thus **S**, on the one hand imply more elements of **S** need to be recovered (harder), yet on the other hand more subtraces per element are available (easier). To complicate matters, the elements of **S** are chosen to be relatively small in $\mathbb{Z}_q$, with the exact support differing per parameter set. All in all, the effect of parameter selection on the natural side-channel resistance is multi-faceted and potentially counterintuitive. I therefore provide guidance in this respect in Section 3.5.

For my investigation, I opted for the ARM Cortex M0 as the platform for Frodo's implementation. The Cortex-M family has high practical relevance in the IoT panorama, where my choice for the M0 is primarily instigated by the availability of the ELMO tool [MOW17], which I use to simulate Frodo's power consumption (see Section 3.2 for details).

Overall, I target up to three points of interest, corresponding to three assembly instructions: loading of a secret value, the actual $\mathbb{Z}_q$ multiplication, and updating an accumulator with the resulting product. For a classical divide-and-conquer attack, where all positions of the secret matrix **S** are attacked independently, the templates can easily be profiled at the start, but as I find in Section 3.3, the resulting algorithmic variance is too high to allow meaningful key recovery.

Therefore I switch to an extend-and-prune technique (Section 3.4), allowing inclusion of predictions on intermediate variables (such as partial sums stored into an accumulator). This approach drastically reduces the algorithmic variance and hence increases the effective signal strength. I show how different pruning strategies allow for a trade-off between performance and success, concluding that for reasonable levels of success, this type of pruning needs to be less aggressive than that employed by Aysu et al. [ATT+18]. I also find that of the two Frodo parameter sets given in the NIST proposal, the one designed for higher black-box security is in fact the most vulnerable against side-channel cryptanalysis.

3.2 Preliminary Notions

The core operation of Frodo is the calculation of $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E}$. Without loss of generality, I will henceforth concentrate on only a single column of the secret matrix **S**, which

will be denoted by $\mathbf{s}$. Thus all attacks in this chapter target the operation $\mathbf{b} \leftarrow \mathbf{As} + \mathbf{e}$, where I try to recover the vector $\mathbf{s}$ for known $\mathbf{A}$ and $\mathbf{b}$ based on the leakage from primarily the matrix–vector multiplication $\mathbf{As}$. Note that, given $\mathbf{A}$ and $\mathbf{b}$, it is possible to check whether a guess $\mathbf{s}$ is correct by checking whether $\mathbf{b} - \mathbf{As}$ is in the support of χ. This suffices with very high probability, because a wrong $\mathbf{s}$ would make the result pseudorandom.

The analysis of a single column recovery $\mathbf{s}$ could easily be extrapolated to the recovery of the full secret matrix $\mathbf{S}$ by taking into account the number of columns $\overline{n}$ and the fact that columns can be attacked independently. Furthermore, for the original Frodo key agreement, a subsequent step in the protocol to arrive at a joint secret, the so-called reconciliation, is component-wise. Consequently, correctly recovering one column of $\mathbf{S}$ immediately translates to recovering part of the eventual session key (between 8 and 32 bits, depending on the selected parameter set). A similar argument applies to the public key encryption scheme on which the KEM variant [NAB+17] is based. However, the introduction of hash functions in the final KEM protocol structurally prevents such a threat and full recovery of $\mathbf{S}$ is required.

While I focus on Frodo's operation $\mathbf{As}$, results apply equally to the transpose operation $\mathbf{s}^\intercal\mathbf{A}$, or indeed to any scenario where a small secret vector is multiplied by a public matrix and there is a method to test (as in the case for LWE) with high probability whether a candidate $\mathbf{s}$ is correct. Moreover, despite the focus being on the parameter sets relevant to Frodo (which has relatively leak-free modular reductions due to its power-of-two modulus q), the techniques apply to other parameter sets used in different LWE-based schemes as well. Chapter 6 contains more details on this aspect of my work.

3.2.1 Matrix–vector multiplication.

Algorithm 11 contains the high level description of textbook matrix–vector multiplication. This is usually deployed since asymptotically faster methods have overheads which makes them unsuitable for the matrix dimensions found in practical lattice-based schemes.

For every iteration of the outer loop, the accumulator sum is initialised to zero and updated n times with as many $\mathbb{Z}_q$ multiplications. This means that for every secret entry s_i an adversary can exploit n portions of the power trace, namely each time it is used in Line 5, motivating the use of a horizontal attack.

Algorithm 11 Matrix–vector multiplication as implemented in Frodo.

Input: $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$; $\mathbf{s}, \mathbf{e} \in \mathbb{Z}_q^n$
Output: $\mathbf{b} \leftarrow \mathbf{As} + \mathbf{e}$

1: $\mathbf{b} \leftarrow \mathbf{e}$
2: **for** $r = 1, \ldots, n$ **do**
3: $\quad sum \leftarrow 0$
4: $\quad$ **for** $i = 1, \ldots, n$ **do**
5: $\quad\quad sum \leftarrow sum + A_{r,i} \cdot s_i$
6: $\quad b_r \leftarrow (b_r + sum) \bmod q$
7: **return b**

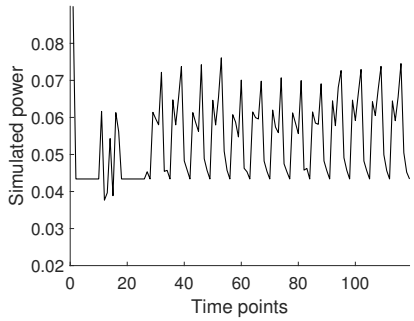
Note that Line 5 does not include an explicit modular reduction. As the modulus q is a power of two, the accumulator sum is allowed to exceed q and will only be reduced modulo q when it is added to the error in Line 6. The modular reduction is really just a truncation, thus does not depend on the value contained in the register holding **b** and therefore I do not exploit its contribution in my analysis.

3.2.2 Experimental Setup

As the target architecture for my experiments I chose the entry level ARM architecture, the Cortex series, because it represents a realistic target and is extremely widely distributed. The Cortex series has several family members, and for the M0 a high quality leakage modelling tool exists. Understanding different attack strategies on different noise levels requires many experiments (I used well over 10^6 full column traces per parameter set), which becomes problematic on real devices. Thus I opted to use simulated *yet realistic* traces which are quicker to generate, modify, and analyse. This allowed me to speed up my analysis, and therefore enable the exploration of a wider noise spectrum.

3.2.2.1 ELMO

ELMO [MOW17] (whose implementation is also public [MOW]) is a tool to simulate instantaneous power consumption for the ARM Cortex M0 processor. This simulator, created by adapting the open-source instruction set emulator Thumbulator [Wel], has been designed to enable side-channel analysis without requiring a hardware measurement setup. ELMO takes ARM thumb assembly as input, and its output describes the power consumption, either at instruction or cycle accuracy. The resulting traces are noise free, that is, they are based deterministically on the instructions and their inputs.



(a) First chunk of power trace as simulated by ELMO of my ARM implementation

Instruction	Operation
`ldrh r5,[r0,r4]`	load s_i
`ldrh r6,[r1,r4]`	load $A_{r,i}$
`muls r5,r6`	$s_i \cdot A_{r,i}$
`adds r3,r3,r5`	$sum + s_i \cdot A_{r,i}$

(b) Breakdown of instructions forming the repeating pattern.

Figure 3.1: Visual representation and detailed structure of target power traces.

ELMO's quality has been established by comparing leakage detection results between simulated and real traces from a STM32F0 Discovery Board [MOW17]. As raw ELMO traces are noise free, the tool is ideal to study the behaviour of template attacks across different noise levels efficiently: both template building and creating noisy traces are straightforward.

Let me stress that ELMO does capture differential data-dependent effects, such as those caused by neighbouring instructions, as well as higher order leakage terms. Consequently, even though ELMO traces are noise free, the trace for the same machine line of code (same operation with the same operand) will differ depending on the context, leading to the already mentioned algorithmic variance, see Section 2.3. Since ELMO's entire model is known and well documented, I treat it as a black box and only query simulated power values by giving as inputs the necessary information.

3.2.2.2 Reference implementation

I implemented the innermost loop of Algorithm 11 in ARM assembly, the code is available in Appendix A, which was wrapped in C code for initialization and loop control. This gives a fine control over the code ELMO simulates the power consumption of and prevents the compiler from inserting redundant instructions which might affect leakage. The code in Appendix A is then just repeated n times because of the outer loop in Algorithm 11.

Figure 3.1a plots a partial power trace of the ARM implementation, as simulated by ELMO. After initialisation, a pattern neatly repeats, corresponding to the equivalent of Line 5 in Algorithm 11. After excluding unimportant points (e.g. loop structure), the most relevant instructions responsible for the pattern are given in Figure 3.1b.

The index i stored in `r4` is used to load values from a row of $\mathbf{A}$ and $\mathbf{s}$, whose addresses are in `r1` and `r0`, respectively, into `r6` and `r5`. These are then used to perform one element multiplication, whose result overwrites `r5`. Finally the accumulator is updated in `r3` and eventually returned.

Negative numbers are wrapped around modulo q. This is in contrast to Frodo's original convention of truncating numbers after 16-bit independently on the parameter set. I expect the higher Hamming weights to amplify leakage, thus making my decision, motivated by simplicity of analysis, very conservative. Finally, intermediate multiplications and partial sums are truncated only when exceeding 32 bits, given the M0 is a 32-bit architecture.

3.2.2.3 Realistic noise estimate

As mentioned before, ELMO traces are noise free. However, when attacking an actual ARM Cortex M0, environmental noise will be introduced. For the experiments, I will artificially add this noise, which I assume independently and identically distributed for all points of interest, according to a normal distribution with mean 0 and variance σ^2.

For the profiling that led to the development of ELMO [MOW17], the observed value[1] of σ was approximately $4 \cdot 10^{-3}$. I will use this realistic level of environmental noise as a benchmark throughout. Furthermore, I will consider a representative range of σ roughly centred around this benchmark. I chose σ in the interval $[10^{-4}, 10^{-2})$ with steps of $5 \cdot 10^{-4}$. Compared to the variance of the signal, my choice implies σ ranges from having essentially no impact to being on the same order of magnitude of the power traces as a whole, thus on the same order of magnitude as the signal variance.

3.2.3 Template attacks

As I briefly described in Section 2.3, a template attack is a powerful technique that builds on the idea that the power model an adversary exploits is built from the device under attack (or an equivalent one) directly. I deferred details until now because the generic formulation is quite heavy in notation and is out of scope for this thesis. Here, instead, I instantiate the relevant formulae for the case of Frodo. As I will briefly show, template attacks apply particularly well in this scenario, as the keyspace is very limited compared to many pre-quantum cryptosystems. Two distinct phases exist.

[1] Personal communication with C. Whitnall.

3.2.3.1 Profiling phase

The goal of this step is to profile the device in order to analyse the effect of different key guesses. In this case, only $\left|\text{Supp}\left(\chi\right)\right|$ (e.g. 23 for NIST1 and 21 for NIST2) possible secret values exist, while the public operand of the $\mathbb{Z}_q$ multiplication is uniformly random but known.

The precise procedure according to which templates are built depends on the attack strategy, as described in Section 2.3.

- In the divide-and-conquer strategy, an adversary builds templates before starting the attack because each subkey, i.e. each position of $\mathbf{S}$, is targeted independently. Moreover, only loading of the secret and multiplication from Line 5 can be targeted. Therefore, it is enough to build $\left|\text{Supp}\left(\chi\right)\right| \cdot q$ templates, i.e. one per each couple $(s,a) \in \text{Supp}\left(\chi\right) \times \mathbb{Z}_q$.

- The extend-and-prune strategy instead builds templates on-the-fly and adaptively, targeting secret positions in order from first to last. This means that, since early positions are recovered first, a guess for the value of the accumulator can also be formulated, therefore including the corresponding addition in the power model. Practically speaking, this means that, when targeting s_i, $\left|\text{Supp}\left(\chi\right)\right| \cdot n$ templates must be built for each candidate vector $(s_1, \ldots, s_{i-1})$, where the factor n comes from the number of elements in a column of $\mathbf{A}$.

Despite having very different approaches, I introduce a common formalism to denote templates. Let $\mathscr{L}(s,c,a) \in \text{Supp}\left(\chi\right) \times \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{R}^T$ be a function that takes as inputs the operands of the operation $a \cdot s + c$ and returns the vector of ELMO voltage values from the corresponding power trace. Since ELMO is a deterministic function, i.e. to same inputs correspond same outputs, those simulated power points are only coming from the contribution of P_{exp}, see Section 2.3. To ease the notation, I call $\mathbf{g}^{s,c,a} = \mathscr{L}(s,c,a) \in \mathbb{R}^T$. In other words, $\mathbf{g}^{s,c,a}$ is a vector of T simulated power points corresponding to the signal an adversary wishes to extract information from. The inputs of $\mathscr{L}$ are the previous value of the accumulator $c \in \mathbb{Z}_q$, which can be just set to zero for divide-and-conquer since it is neglected, a public operand $a \in \mathbb{Z}_q$ and a secret operand $s \in \text{Supp}\left(\chi\right)$. I adopt the well known assumption [MOP07, Section 4.4] that noise follows a multivariate Gaussian distribution. Thus the target power trace of $a \cdot s + c$ is modelled as a random variable $\mathbf{t}^{s,c,a} \in \mathbb{R}^T$ such that

$$\mathbf{t}^{s,c,a} \sim \mathscr{N}_T\left(\mathbf{g}^{s,c,a}, \Sigma^{s,c,a}\right) .$$

In other words, a template is a statistical model that I assume is shaped as a multivariate Gaussian distribution, whose mean is the contribution of the signal to the power trace and whose variance is that of the noise contribution. Both these quantities are precisely what is profiled from the device in the possession of the adversary.

3.2.3.2 Matching phase

Once templates have been built, an adversary computes a likelihood for each template and estimates which one better models the target power traces. Being the templates built on different key candidates, this translates to a posterior probability distribution on the key space. This is achieved by applying Bayes' theorem. Recall that $\mathbf{t}^{s,c,a}$ is the statistical model of the power trace of $a \cdot s + c$, I denote by $\mathbf{T}^{s,c}$ the matrix whose rows are the $\mathbf{t}^{s,c,a}$ for all a in a column of $\mathbf{A}$. Then for a guess $k \in \mathrm{Supp}(\chi)$

$$\Pr\left[k \,\middle|\, \mathbf{T}^{s,c}\right] = \frac{(\prod_a f(\mathbf{t}^{s,c,a} \mid k))\Pr[k]}{\sum_{\ell \in \mathrm{Supp}(\chi)}(\prod_a f(\mathbf{t}^{s,c,a} \mid \ell))\Pr[\ell]} \tag{3.1}$$

where

$$f\left(\mathbf{t}^{s,c,a} \mid \ell\right) = \Phi_{(\mathbf{g}^{\ell,c,a}, \Sigma^{\ell,c,a})}(\mathbf{t}^{s,c,a}) \,.$$

In essence, templates are Gaussian distributions, while the collected power traces, or their points of interest thereof, are samples. Bayes' theorem is therefore used to derive how likely it is for a power point to be drawn from the distribution described by each template. Since there is one per key candidate, the most likely distribution immediately yields a most likely candidate.

Both formalisms of the profiling and matching phases are fairly standard notions, which can be found in the book by Mangard, Oswald and Popp [MOP07, Section 5.3].

3.2.3.3 Simplification of covariance matrix

Working with a full covariance matrix is tricky, as the formulae require its inverse to be calculated, which is a delicate task because of potential numerical instability problems. Therefore, I adopt a sequence of assumptions to simplify the model. I assume the noise distribution does not depend on the data being processed, i.e. $\Sigma^{c,a,s} = \Sigma$. To simplify computations and avoid numerical instability when matching templates with target power traces, I further assume that covariances are equal to zero. This means that the covariance matrix Σ is a diagonal matrix whose diagonal is filled with $\sigma_1^2, \ldots, \sigma_T^2$, where T is the number of time points included in the power trace.

Overall, such assumptions do not strictly hold in practice, but they have been extensively studied in the literature [CK14], [MOP07, Section 5.3], leading to the conclusion that their associated numerical benefits outweigh the loss of precision due to model errors.

3.2.3.4 Simplification of the distinguisher

The description of template attacks, particularly of the matching phase, showed formal details on the mathematics behind the application of templates. If one was to work with those formulae directly, however, efficiency and numerical stability problems would arise. The inversion of a matrix with small entries and fractions involving very small numbers require much care. The final goal of a template attack, however, is not strictly to derive a posterior probability distribution over the key space, as Bayes' theorem allows one to do, but to decide which key candidate is the most likely in use by the device under attack. Bayes' formulae overshoot at such a goal, which can be achieved with several well known simplifications.

First of all, recall that I modelled a power trace as

$$\mathbf{t}^{s,c,a} \sim \mathcal{N}_T\left(\mathbf{g}^{s,c,a}, \Sigma\right) .$$

where $\mathbf{g}^{s,c,a}$ is the part of interest to the adversary, the true signal which only depends on its inputs. In light of this, I rewrite the above as follows

$$\mathbf{t}^{s,c,a} = \mathbf{g}^{s,c,a} + \mathbf{N}_a$$

where $\mathbf{N}_a \sim \mathcal{N}_T(\mathbf{0}_T, \Sigma)$ represents the noise component. The index a simply means the vector of noise values corresponding to the computation when $a \in \mathbb{Z}_q$ is used. Noise values themselves, however, are independent of a.

Now I start from Equation (3.1) and simplify away all terms that do not depend on the key candidate k. Indeed, additive and multiplicative factors which are applied to all candidates do not essentially change their order, hence whichever had the highest probability before the transformations will have the highest *score* after them. First of all, I can neglect the denominator as it is simply a scaling factor equally applied to all candidates. Its formulation does not indeed depend on k. Applying a monotonic function also does not change the orders of score, hence taking the logarithm is allowed. After

these two transformations, scores look like

$$\begin{aligned}\tilde{S}(k \mid \mathbf{T}^{s,c}) &= \ln\left(\prod_a f\left(\mathbf{t}^{s,c,a} \mid k\right)\right) + \ln \Pr[k] \\ &= \sum_a \ln\left(f\left(\mathbf{t}^{s,c,a} \mid k\right)\right) + \ln \Pr[k] \ .\end{aligned}$$

By applying the definition of f and of the probability density function, the above becomes

$$\begin{aligned}\tilde{S}(k \mid \mathbf{T}^{s,c}) &= \sum_a \left(\ln\left(\frac{1}{\sqrt{\det(2\pi\Sigma)}} \exp\left(-\frac{1}{2} \cdot (\mathbf{t}^{s,c,a} - \mathbf{g}^{k,c,a})\Sigma^{-1}(\mathbf{t}^{s,c,a} - \mathbf{g}^{k,c,a})^\intercal\right)\right)\right) + \ln \Pr[k] \\ &= \sum_a \left(\ln\left(\frac{1}{\sqrt{\det(2\pi\Sigma)}}\right) - \frac{1}{2} \cdot (\mathbf{t}^{s,c,a} - \mathbf{g}^{k,c,a})\Sigma^{-1}(\mathbf{t}^{s,c,a} - \mathbf{g}^{k,c,a})^\intercal\right) + \ln \Pr[k] \\ &= \sum_a \ln\left(\frac{1}{\sqrt{\det(2\pi\Sigma)}}\right) - \frac{1}{2}\sum_a \left((\mathbf{t}^{s,c,a} - \mathbf{g}^{k,c,a})\Sigma^{-1}(\mathbf{t}^{s,c,a} - \mathbf{g}^{k,c,a})^\intercal\right) + \ln \Pr[k]\end{aligned}$$

The first term is a constant for all candidates k, therefore the final definition of scores is

$$S(k \mid \mathbf{T}^{s,c}) = \ln \Pr[k] - \frac{1}{2}\sum_a (\mathbf{t}^{s,c,a} - \mathbf{g}^{k,c,a})\Sigma^{-1}(\mathbf{t}^{s,c,a} - \mathbf{g}^{k,c,a})^\intercal$$

which can be rewritten as

$$S(k \mid \mathbf{T}^{s,c}) = \ln \Pr[k] - \frac{1}{2}\sum_a \sum_{i \le T} \frac{(t_i^{s,c,a} - g_i^{k,c,a})^2}{\sigma_i^2} \ . \tag{3.2}$$

To recap, the term $\Pr[k]$ is simply $\chi(k)$; the summation over a indicates the use of n power traces, one per element in a column of $\mathbf{A}$; the last summation using i as index ranges over all time points considered in the attack. This fact will be made explicit in the next section.

3.3 Divide-and-Conquer Template Attack

As every entry of $\mathbf{s}$ is an independently and identically distributed sample from χ, I can potentially target each position separately. Thus I first consider a divide-and-conquer template attack. A distinct advantage of this approach is that the total number of templates is fairly small and hence profiling can be preprocessed.

When considering the breakdown of the inner loop (Figure 3.1b), I ignore the loading of the public operand (it essentially leaks nothing exploitable), which leaves three potential points of interest. On the one hand, the loading of the secret operand and the multiplication contain direct leakage on the secret, and all relevant inputs appear

known. For the accumulator update on the other hand, the leakage is less direct and the value of the accumulator so far cannot be taken into account: it depends on the computation so far, violating the independence requirement for divide-and-conquer. Thus, the attack in this section is limited to *two* points of interest, namely the loading of the secret and the $\mathbb{Z}_q$ multiplication. From the point of view of the formalism defined in Section 3.2, this means that the parameter c in $\mathbf{T}^{s,c}$ can simply be ignored (and fixed to 0) because it does not affect the two chosen points of interest. When this is the case, I will write $\mathbf{T}^{s}$. I will give more details on this aspect in Section 3.3.4.

Of course, one could still generate templates for all *three* points of interest by treating the accumulator as a random variable. However, as the accumulator value is a direct input to the accumulator update and its register is used for the output as well, the resulting algorithmic variance would be overwhelming. Indeed, as I will show below, already for the loading of the secret there is considerable algorithmic variance related to the previous value held by the relevant register. These limitations are intrinsic to a divide-and-conquer approach; in Section 3.4 I show how an extend-and-prune approach bypasses these problems.

3.3.1 Estimating success rates

For each entry s_i, the distinguisher outputs a distinguishing score vector that can be linked back to a perceived posterior distribution. Selecting the element corresponding to the highest score corresponds to the *Maximum A Posteriori* (MAP) estimate and the probability that the correct value is returned this way is referred to as the first-order success rate.

Ultimately, I am more interested in the first order success rate of the full vector $\mathbf{s}$. As I assume independence for a divide-and-conquer, I can easily extrapolate the success rates for $\mathbf{s}$ based on those for individual positions as a full vector is recovered correctly if and only if all its constituent positions are. The advantage of using extrapolated success rates for $\mathbf{s}$, rather than using direct sample means, is that it provides useful estimates even for very small success rates (that would otherwise require an exorbitant number of samples). Thus, analysing the recovery rates of single positions is extremely informative. Additionally, it gives insights on why the extend-and-prune attack in Section 3.4 greatly outperforms divide-and-conquer.

Other metrics, beyond first-order recovery rate, are of course possible to compare between distinguishers [SMY09]. However, I regard those alternatives, such as oth-order recovery or more general key ranking, only of interest when first order success

rate is low. While for divide-and-conquer this might be the case, for extend-and-prune the first order recovery is sufficiently high to warrant concentrating on that metric only.

3.3.1.1 Estimating position success rate

Let $\Pr[S]$ be the first order position recovery rate where S is the event that the distinguisher indeed associates the highest score to the actual secret value. I experimentally estimate $\Pr[S]$ based on the formula

$$\Pr[S] = \sum_{s \in \mathrm{Supp}(\chi)} \Pr[S \mid s] \Pr[s]$$

where $\Pr[s]$ corresponds to the prior distribution χ and the values for $\Pr[S \mid s]$ are estimated by appropriate sample means. To ensure the traces are representative, I range over $\mathbf{A}$ and $\mathbf{s}$ (and $\mathbf{e}$) for the relevant experiments and generate traces for the *full* computation $\mathbf{b} \leftarrow \mathbf{As} + \mathbf{e}$. This allows me to zoom in on individual positions, highlighting where algorithmic variance occurs. While one could also use direct, position-specific sample means for $\Pr[S]$, my approach links more closely to the confusion matrix and has the advantage that it depends less on the sampling distribution of $\mathbf{s}$ when running experiments.

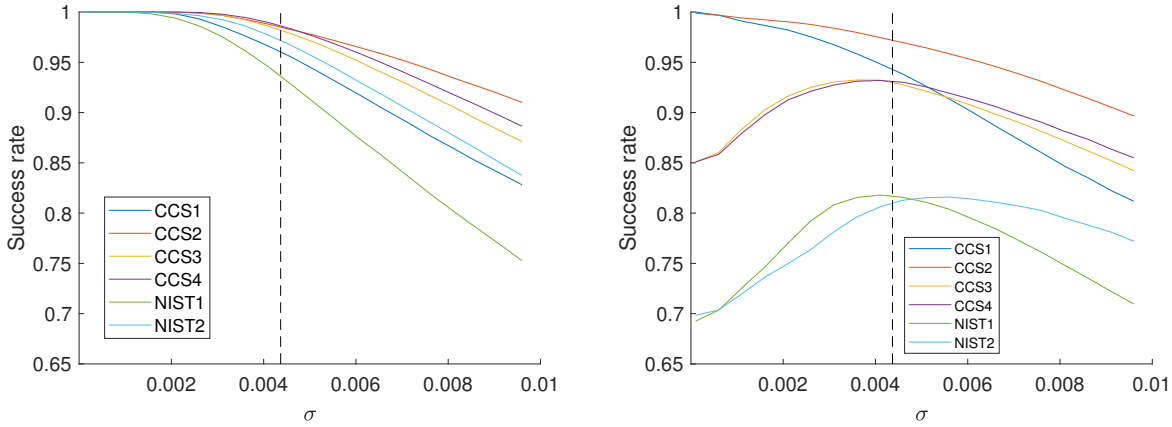
3.3.1.2 Extrapolating overall success rate

If I assume independence of positions, it is easy to express the overall success rate for recovering $\mathbf{s}$. If I, temporarily, make the simplifying assumption that $\Pr[S]$ is the same for all n positions, then the first order recovery rate for $\mathbf{s}$ is $\Pr[S]^n$ (recovery of $\mathbf{s}$ will be successful if and only if recovery of each of its elements is). Even for extremely high $\Pr[S]$, this value quickly drops, e.g. $0.99^n \approx 5.5 \cdot 10^{-5}$ for NIST2.

3.3.2 Results with basic distinguisher

The first set of experiments I propose comes from my original publication at the SAC conference, where I chose to simplify the divide-and-conquer approach in favour of a deeper analysis of extend-and-prune, here described in Section 3.4.

As I have already mentioned, the only points of interest considered in this section are the loading of the secret and multiplication. Furthermore, I take a step forward in the simplification of Equation (3.1) by assuming that the covariance matrix Σ is not only diagonal, but also with fixed variances. In other words, $\Sigma = \sigma^2 \mathbf{I}_2$. As I will show



(a) Recovery rate for first position only.

(b) Recovery rate for second position only.

Figure 3.2: Comparison of recovery rates between first and second positions. The dashed black line indicates my choice of realistic noise level.

in Section 3.3.3, this a first approximation that does not hold in general, however it simplifies the scores in Equation (3.2) to

$$S(k \mid \mathbf{T}^s) = \ln \Pr[k] - \frac{1}{2\sigma^2} \sum_a \sum_{i \le 2} (t_i^{s,a} - g_i^{k,a})^2 \ .$$

Note that I have already specialised $T = 2$, i.e. two points of interest, and that I dropped any reference to the parameter c.

Being it an approximation, it introduces a certain degree of error, which depends on the operation. For the sake of keeping the computations simple, I ignore this fact for this distinguisher (thus the name "basic"). This is done in order to simplify away the contribution of algorithmic variance and, more in general, of any cross-dependency between points. In other words, such cross-dependencies need not to be estimated now. I defer an in-depth analysis to Section 3.3.3, where I show how including algorithmic variance in the templates improve success rate.

3.3.2.1 Experimental results

I target each position of $\mathbf{s}$ individually, but only report on the first and second one. Figure 3.2 displays the success rate for all parameter sets. Each point in each curve is based on $8 \cdot 10^5$ experiments. The left panel (Figure 3.2a) plots the success rate for the first position, whereas the right panel (Figure 3.2b) plots it for the second position. The x axis ranges over the noise interval I discussed in Section 3.2.2.3. In other words,

templates work in a very low noise regime on the left and in a very high noise regime on the right.

3.3.2.2 The impact of algorithmic variance

The striking difference between Figures 3.2a and 3.2b, especially in the low environmental noise regime, is due to algorithmic variance. As I mentioned before, algorithmic variance particularly affects the loading of the secret, i.e. the first instruction in Figure 3.1b, due to the previous register value contributing to the leakage. This problem only appears from the second position onward; for the first position, no algorithmic variance is present as the initial state is fixed and known. This makes Figure 3.2b representative of all subsequent positions.

With the exception for the two small CCS parameter sets, even with virtually no environmental noise, the success rate for the second position is far from 1. Moreover, when environmental noise is added, the success rate initially goes up. This phenomenon is known as stochastic resonance [MSPA08] and has been observed for side-channels before [WO11]. Even for CCS1 and CCS2, that have the lowest algorithmic variance level, the success rate for the second position is slightly lower than for the first position.

For completeness, the assumption that the noise covariance matrix Σ for my two points of interest is a diagonal matrix $\sigma \cdot \mathbf{I}_2$, is suboptimal in the presence of algorithmic variance. Using a diagonal matrix Σ that incorporates the algorithmic variance would improve the distinguisher while reducing the stochastic resonance. I will explore this case in more detail in Section 3.3.3.

3.3.2.3 Full vector recovery

The success rates for full vectors are more relevant to compare either amongst parameter sets or with other attacks, be it lattice or other side-channel attacks. As a simplification, I assume that the recovery rate for the second position (Figure 3.2b) is representative for all positions bar the first, whose contribution is limited anyway given concrete values of n (the total number of positions). This is a fair assumption to make because the first position does differ from all the others, because registers and preceding instructions are independent from the secret and can therefore be taken into account in the templating phase.

To ease comparison, for each parameter set I determined the σ for which the divide-and-conquer attack approximately achieves a success rate for recovering $\mathbf{s}$ of around 2^{-128} (corresponding to 128-bit security). For the smallest parameter sets, CCS1 and

CCS2, all the σ in the range are susceptible (i.e. lead to success rates of at least 2^{-128}), whereas for the NIST parameter sets, none of the σ appear insecure. For the original large sets CCS3 and CCS4, any σ below $7 \cdot 10^{-3}$, which includes my realistic benchmark, leads to a loss of security below the 128-bit level.

I acknowledge that a further reduction in residual bit security might be possible by considering key ranking, or possibly even novel lattice reduction algorithms that take into account side-channel information. To the best of my knowledge, none of these approaches have been applied in the context of LWE schemes and therefore I cannot comment on their efficacy.

3.3.3 Including algorithmic variance to improve the distinguisher

The root cause of algorithmic variance generating the stochastic resonance observable in Figure 3.2b lies in the inputs given to ELMO during the generation of target power traces but that cannot be given when templates are generated. This is because, to simulate a real device, target power traces are simulated using whole rows of $\mathbf{A}$ and columns of $\mathbf{S}$, while templates profile specific $\mathbb{Z}_q$ multiplications only. An adversary only has a restricted view in the latter procedure, namely of only one secret position at the time. This does not mean, however, that a smart adversary cannot include knowledge of the device (of ELMO, in this case) into the templates, and build them accordingly. More details about the functioning of ELMO are needed to explore this idea.

Each point in a power trace generated by ELMO is a deterministic function of a triplet encoding previous, current and subsequent instructions; and two couples of operands, one for the previous and one for the current instructions. This already suggests why during template generation some information is missing: when targeting any $i \neq 1$, an adversary does not know all the operands.

More precisely, with reference to Figure 3.1b, the operands for the three listed types of instruction are as follows, where I denote by $\mathbf{a}$ and $\mathbf{s}$ the two vectors from $\mathbf{A}$ and $\mathbf{S}$, respectively, being multiplied. I denote the vector of multiplications by $\mathbf{m}$, i.e. $m_i = s_i \cdot a_i$.

- The two operands of `ldrh` instructions are the value previously stored and the one overwriting it, therefore I write $a_i = \texttt{ldrh}(a_{i-1}, a_i)$ for e.g. the loading of the public multiplicand.

- The `muls` instruction simply has its two inputs as operands, i.e. $m_i = \texttt{muls}(s_i, a_i)$.

Note that the output overwrites the register of the first operands, hence `r5` in this case. For this reason loading of the secret is $s_i = \texttt{ldrh}(m_{i-1}, s_i)$.

- The `add` instruction uses the value previously stored in `r3` and what is in `r5` as operands. As I mentioned before, I do not use this instruction for the attack.

In a divide-and-conquer approach, one wants to be able to parallelise and pre-process templates and there is only a limited amount of information one can include in them, hence giving rise to algorithmic variance. Everything that cannot be captured in such parallelisable fashion, despite being related to other parts of $\mathbf{s}$ itself, degrades the quality of templates: adversaries can profile a position including only what is in their view for that position, secret values of other positions must necessarily be left out.

The previous value held by register `r5` when loading a secret is not known (nor guessable for the above reason) by an adversary, as well as the previous value in `r3` when adding to the accumulator. These two points will therefore show algorithmic variance. I ignore the contribution of the instruction preceding the first `ldrh` because it is part of the loop structure and can be fully profiled.

A different discussion applies to multiplication. Theoretically speaking, operands of the previous instruction, i.e. loading of a_i, have nothing unknown. However, an a priori profiling of all combinations of a_{i-1}, a_i and s_i would require kq^2 templates. Such a number ranges between $\approx 2^{25}$ for CCS1 and $\approx 2^{37}$ for NIST1. Despite being within feasible reach, these numbers undermine the practicality of any attack, given how intense profiling is. Fixing $a_{i-1} = 0$, for instance, would be a solution but would inherently introduce some algorithmic variance. I overcome the issue by treating a_{i-1} as unknown, and estimating algorithmic variance on multiplication accordingly.

Remark 3.1. My analysis strictly follows the implementation reported in Figure 3.1b. I acknowledge that there are alternatives which would keep the functionality but would slightly change leakage: swapping the two `ldrh` instructions, swapping operands of the `muls` instruction, swapping operands of the the `add` instruction and any combination thereof. The analysis would be extremely similar to that reported here, therefore I ignore such alterations.

I will now describe both points of interest and how the acquired knowledge can be used to improve the performance of the divide-and-conquer distinguisher.

3.3.3.1 Loading of the Secret

I denote by $\mathrm{ELMO}_{\texttt{ldrh}}(s_{i-1}, a_{i-1}, s_i) \in \mathbb{R}$ the power value produced by ELMO when computing $\texttt{ldrh}(s_{i-1} \cdot a_{i-1}, s_i)$. For all $a \in \mathbb{Z}_q$ and $s \in \mathrm{Supp}(\chi)$, I build the mean of the template as

$$\mu_{\texttt{ldrh}}(a,s) = \sum_{r \in \mathrm{Supp}(\chi)} \chi(r) \cdot \mathrm{ELMO}_{\texttt{ldrh}}(r,a,s) \ .$$

Since a_{i-1} is known and guesses are needed for all possibilities of s_i, the above formula covers all combinations. The algorithmic variance derived from not knowing s_{i-1} is computed as

$$\sigma^2_{\texttt{ldrh}}(a,s) = \sum_{r \in \mathrm{Supp}(\chi)} \chi(r) \cdot (\mathrm{ELMO}_{\texttt{ldrh}}(r,a,s) - \mu_{\texttt{ldrh}}(a,s))^2 \ . \tag{3.3}$$

One thing to be noticed is that the above turns out to be essentially independent of s. In other words $\sigma^2_{\texttt{ldrh}}(a,s) \approx \sigma^2_{\texttt{ldrh}}(a,s')$ for all $s, s' \in \mathrm{Supp}(\chi)$ up to an error between 10^{-33} and 10^{-20}. I therefore denote the right hand side of Equation (3.3) by $\sigma^2_{\texttt{ldrh}}(a)$ from now on.

3.3.3.2 Multiplication

Applying a formal definition of expected value and variance in this case is cumbersome, as doing so would mean estimating all possible combinations of two values in $\mathbb{Z}_q$ and one in $\mathrm{Supp}(\chi)$. These are kq^2, which is a feasible but uncomfortable number to enumerate. Moreover, a simpler yet accurate approach exists: I draw values from the correct distributions and compute sample mean/variance.

Similarly to the above, I denote by $\mathrm{ELMO}_{\texttt{muls}}(a_{i-1}, s_i, a_i) \in \mathbb{R}$ the power value produced by ELMO when computing $\texttt{muls}(s_i, a_i)$. Note that a_{i-1} is needed in input because it is an operand of the preceding instruction, hence does influence the power value. For each couple $(a,s) \in \mathbb{Z}_q \times \mathrm{Supp}(\chi)$, I draw $\alpha = 2^{16}$ values $a_1, \ldots, a_\alpha$ uniformly at random from $\mathbb{Z}_q$. These represent a random selection of values that a_{i-1} can take. I then compute the sample mean of the template as

$$\mu_{\texttt{muls}}(a,s) = \frac{1}{\alpha} \sum_{j \le \alpha} \mathrm{ELMO}_{\texttt{muls}}(a_j, s, a)$$

and the algorithmic variance as

$$\sigma^2_{\texttt{muls}}(a,s) = \frac{1}{\alpha} \sum_{j \le \alpha} (\mathrm{ELMO}_{\texttt{muls}}(a_j, s, a) - \mu_{\texttt{muls}}(a,s))^2 \ . \tag{3.4}$$

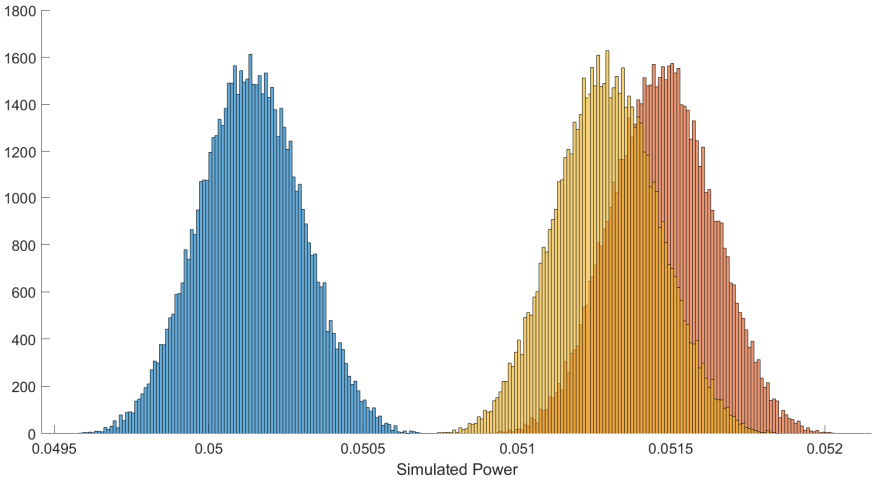


Figure 3.3: Histograms of power values when multiplication between $a_i = 0$ and three values for s_i (0, 1 and 2; blue, red and yellow respectively). I drew $\alpha = 2^{16}$ values uniformly at random from $\mathbb{Z}_q$ for a_{i-1}.

Similarly to the above case, algorithmic variance is essentially independent of a. In other words $\sigma^2_{\texttt{muls}}(a,s) \approx \sigma^2_{\texttt{muls}}(a',s)$ for all $a,a' \in \mathbb{Z}_q$ up to an error between 10^{-24} and 10^{-21}. I therefore denote the right hand side of Equation (3.4) by $\sigma^2_{\texttt{muls}}(s)$ from now on.

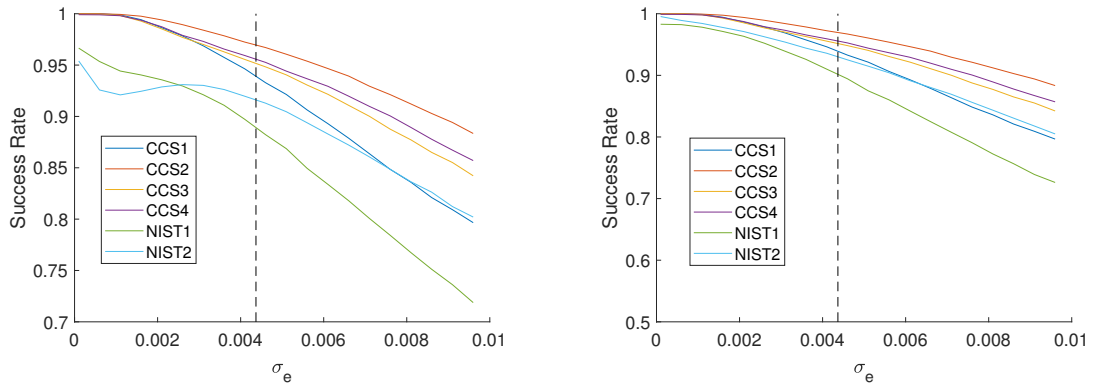
Figure 3.3 shows three histogram plots of power values when $a_i = 0$, s_i assumes three values (0, 1 and 2; blue, red and yellow respectively) and a_{i-1} is drawn uniformly at random. There are 2^{16} samples for the latter, while the values of s_i affect the mean of each distribution. Note that all histograms have almost identical shapes, therefore the underlying distributions have the same variance.

3.3.3.3 Improving the distinguisher and results

Equipped with the templates built as described above, in both their mean and algorithmic variance components, I can apply them in the matching phase of a template attack against ELMO traces.

I denote by $t_i^{\texttt{ldrh}}$ and $t_i^{\texttt{muls}}$ the points in the target power trace computing the ith multiplication. I also assume that environmental Gaussian noise is introduced, with mean 0 and variance σ_e^2. In other words,

$$t_i^{\texttt{ldrh}} = \text{ELMO}_{\texttt{ldrh}}(s_{i-1}, a_{i-1}, s_i) + \mathcal{N}(0, \sigma_e^2)$$
$$t_i^{\texttt{muls}} = \text{ELMO}_{\texttt{muls}}(a_{i-1}, s_i, a_i) + \mathcal{N}(0, \sigma_e^2)\ .$$



(a) Success rate of template attacks including algorithmic variance and environmental noise.

(b) Success rate with the further inclusion of "ghost" variance.

Figure 3.4: Comparison between success rates.

The first position is somewhat special because registers are pre-filled with constants. An adversary can simply take care of this in the templates.

I use the distinguisher in Equation (3.2), this time without assuming that all variances are equal. Therefore, the score of a guess $k \in \mathrm{Supp}(\chi)$ is defined as

$$S(k \mid \mathbf{T}) = \ln \Pr[k] - \sum_{i \le n} \frac{(t_i^{\mathtt{ldrh}} - \mu_{\mathtt{ldrh}}(a_i,k))^2}{\sigma_e^2 + \sigma^2_{\mathtt{ldrh}}(a_i)} + \frac{(t_i^{\mathtt{muls}} - \mu_{\mathtt{muls}}(a_i,k))^2}{\sigma_e^2 + \sigma^2_{\mathtt{muls}}(k)}$$

where $\mathbf{T}$ is the matrix whose rows are $\mathbf{t}^{\mathtt{ldrh}}$ and $\mathbf{t}^{\mathtt{muls}}$. Compared to Equation (3.2), the summation over the time points has been made explicit by expressing the only two points of interest, while the summation over different values for a has been replaced by that with index i, ranging over $\mathbf{a}$.

I applied the distinguisher to 30 seeds, each seed being a different $(\mathbf{A},\mathbf{S})$ couple. Hence I effectively mounted 240 full column attacks, as there are 8 columns in each $\mathbf{S}$. I repeated them 20 times for 20 different values of σ_e. It ranged in $[10^{-4}, 10^{-2})$ with steps of $5 \cdot 10^{-4}$. Figure 3.4a plots position recovery for all parameter sets. All positions have been included, hence no distinction between first and others is made. As usual, the vertical dashed line represent the "realistic" noise level.

Including algorithmic variance in the templates and as part of the distinguisher has improved success rate considerably for all parameter sets. Stochastic resonance has entirely disappeared from the CCS3 and CCS4 parameter sets. However, it is still present in the NIST1 and NIST2 parameter sets, although much less pronounced. This is due to the fact that, as hinted in several places above, also the definition adopted for algorithmic variance in the two points of interest represent an approximation of reality.

Including all possible information in a mathematically sound formula and estimating all templates for all possible values would be problematic. For the sake of filling the gap, I experimentally derived a variance, which I call "ghost" variance, that, if added to the denominators of the distinguisher in the case of NIST1 and NIST2, produces Figure 3.4b. Such a "ghost" variance is $\sigma_g^2 = 0.0053^2$. Stochastic resonance has now completely disappeared and success rate for those two parameter sets has slightly improved across all σs.

The two graphs of Figure 3.4 are asymptotically similar, particularly so from the "realistic" noise level on. This means that the "ghost" variance affects results only with low environmental noise, which is not an interesting scenario when utilising traces from a real device. The result achieved by Figure 3.4a is therefore sound and satisfactory.

3.3.4 The role of the accumulator

I conclude my study of the divide-and-conquer strategy by exploring why it is indeed beneficial to exclude the addition with the accumulator from the points of interest. For that point to be of any interest to an adversary, it would need to contain more signal about the current secret position under attack than noise. The problem is that the value of the accumulator before the addition does not belong to the adversary's view and contributes, therefore, to the algorithmic noise on that point of interest.

In practical terms, for a fixed value of the public operand and a uniformly random value of the accumulator, distributions of power values when different secret values are involved should be separable enough (where by "enough" I mean given the limited trace chunks available). Figure 3.5 shows precisely such a scenario for the NIST1 parameter set: each overlapping histogram is formed of ELMO power values corresponding to the addition with the accumulator. As I mentioned above, there are three variables that contribute to the computation: the public operand, which is known and fixed to a non-zero value, the secret operand, which is a different one for each histogram in Figure 3.5, and the value of the accumulator. To simulate an adversary not knowing the latter, I drew 2^{16} uniformly random values from $\mathbb{Z}_{2^{32}}$, because it can be any 32-bit number. What Figure 3.5 narrates is twofold: first of all the signal, i.e. the contribution of the secret element, is in the mean of the distributions; however, means for several secrets are so close to each other and variances, which is worth noticing are fairly independent of the secret, are so wide that differences are hard to identify. Only in some corner cases this would be doable, e.g. the left-most lightblue distribution corresponding to the value 1 against the right-most purple distribution corresponding to the value $2^{15} - 1$. This is

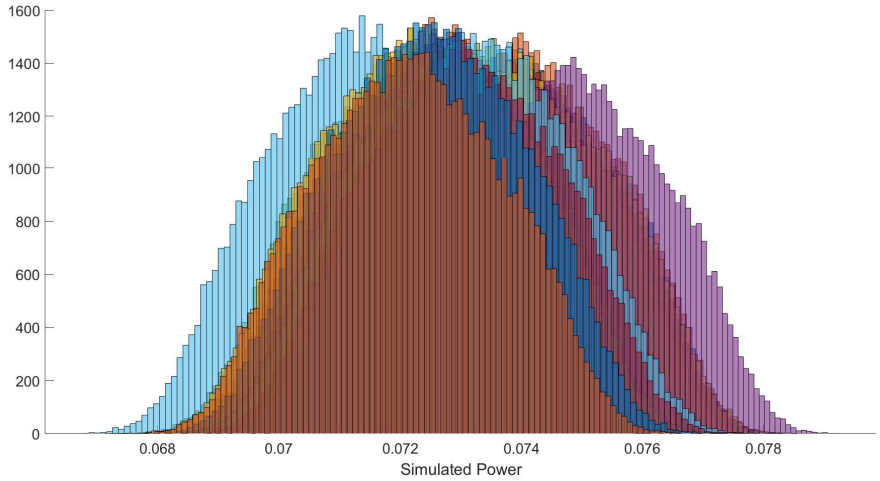


Figure 3.5: Histograms of power values when addition with accumulator is performed for the NIST1 parameter set. Each histogram refers to a different secret value in $\mathrm{Supp}(\chi)$, the value of the public operand is fixed to a non-zero value at random, while 2^{16} samples are drawn uniformly at random from $\mathbb{Z}_{2^{32}}$, to simulate the 32-bit random value of the accumulator.

especially true since telling distributions apart is a task requiring more traces the more mixed together they are, and the setting considered in this chapter greatly limits it.

3.4 Extend-and-Prune Template Attack

For the divide-and-conquer approach from the previous section, I assumed that the positions of $\mathbf{s}$ are independent of each other. While this assumption is valid for the generation of $\mathbf{s}$, it turned out that for the leakage, it is not. However, Algorithm 11 deals with the elements of $\mathbf{s}$ *sequentially*, from position 1 to position n, which I will exploit by a well-known extend-and-prune approach.

In this case, the extend-and-prune algorithm operates as follows. I imagine a k-ary tree of depth n where the nodes at level i in the tree correspond to a partial guess $s_1, \ldots, s_{i-1}$ for the secret; for a given node at level i, its k out-going edges are labelled by the k possible values that s_i can take. This way, each path from the root to one of the k^n possible leaves uniquely corresponds to one of the possible values that the secret vector $\mathbf{s}$ can take. A distinguisher can sequentially calculate a score for a vector $\mathbf{s}$ by traversing the tree from the root to the leaf representing $\mathbf{s}$ where for each edge it encounters, it cumulatively updates $\mathbf{s}$'s score.

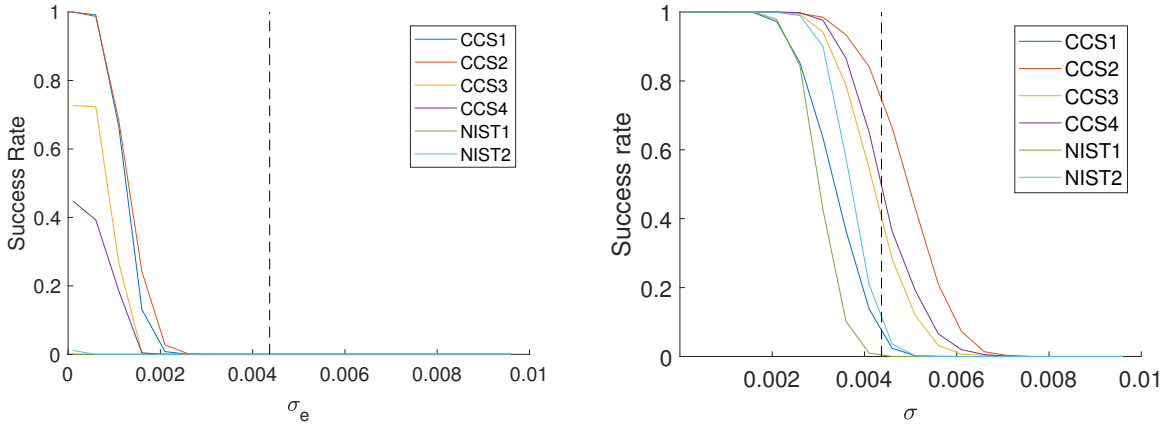
The challenge of an extend-and-prune algorithm is to efficiently traverse a large tree (as big as the key space) while still ending up with a good overall score. The standard way [CRR03] of doing so is to first calculate the score for all nodes at level 2. For each level-2 node, the score will be that of the edge from the root to that node. Thus the trivial level-1 guess is *extended* to all possible level-2 guesses. The next stage is to *prune* all these guesses to a more reasonable number. For all the remaining level-2 guesses, one then extends to all possible level-3 guesses, and then again these guesses are pruned down. Such a process repeats until reaching the final level $(n+1)$, where the complete $\mathbf{s}$ is guessed.

The advantage of this approach is that, when calculating a score for s_i, the distinguisher already has a guess for $s_1,\ldots,s_{i-1}$, which allows it to create templates based on this guess. My distinguisher will only use the previous secret s_{i-1} and the value of the accumulator so far (an inner product of $(s_1,\ldots,s_{i-1})$ with the relevant part of $\mathbf{A}$) to create a template. As the total number of possible templates becomes rather unwieldy (around $k^2 \cdot q \cdot 2^{32}$), the profiling is interleaved with the tree traversal and pruning is used to keep the number of templates manageable.

The success of an extend-and-prune attack depends on the pruning strategy, specifically how many candidates to keep at each step. To the best of my knowledge, there is no comprehensive study comparing different pruning strategies in different scenarios. When Chari et al. [CRR03] introduced template attacks to the cryptanalyst's arsenal, they suggested a pruning strategy that depends on the scores themselves. I instead fix the same number of candidates to keep at each step, which is a classical approach known as *beam search* [Red77]. The size of the beam, that is the number of candidates to keep after pruning, is denoted by b.

3.4.1 Greedy pruning using a laser beam ($b = 1$)

I start by considering the greediest pruning strategy by restricting the beam size to $b = 1$, meaning that after each step I only keep a single candidate, that with the highest score, for the secret recovered so far. This "knowledge", provided it is correct, has two very immediate effects. Firstly, the algorithmic variance I observed in the loading of the secret can be reduced as I assume I typically know the previous secret held by the relevant register. Secondly, by recovering $\mathbf{s}$ from first to last I can predict the value of the accumulator, which brings into play a third point of interest, namely the update of the accumulator (the last point in Figure 3.1b), as here too the algorithmic variance disappears.



(a) Column recovery rate of divide-and-conquer.

(b) Column recovery rate of extend-and-prune.

Figure 3.6: Comparison between column recovery of my two template attacks.

Figure 3.6 presents the vector recovery rates of both divide-and-conquer attack with improved distinguisher, see Section 3.3.3 (in the left panel, Figure 3.6a), and of extend-and-prune using $b = 1$ (Figure 3.6b). The former is extrapolated based on position recovery rates from Figure 3.4b, whereas the latter has been estimated directly, based on $2 \cdot 10^3$ experiments per setting.

The difference between Figures 3.6a and 3.6b is striking. For the extend-and-prune approach I almost completely removed algorithmic variance and, when virtually no environmental noise is present either ($\sigma \approx 10^{-4}$), this resulted in a vector recovery rate of essentially 1. However, when considering the realistic noise level as indicated by the dashed vertical line, not all parameter sets are as affected. A simple modification to the pruning strategy, however, makes sure that little hope remains.

3.4.2 Increasing the beam size ($b > 1$)

So far I only considered $b = 1$. Increasing the beam size b will result in a slower key recovery (linear slowdown in b) but should yield higher recovery rates. For $b = 1$ I mentioned two advantages of extend-and-prune, namely reduced algorithmic variance and an additional point of interest. For $b > 1$ a third advantage appears, namely the ability for the distinguisher to self-correct. This self-correcting behaviour has also been observed (for the first position) by Aysu et al. [ATT+18], who essentially used a beam size $b > 1$ for the first position and then revert to $b = 1$ for all remaining ones.

To assess the effect of the beam size b, I ran two types of experiments. Firstly, for

Name	$b_{\min}$	b								
		2	3	4	5	6	7	8	9	10
CCS1	30709	0	0	0	0	0	0	0	0	0
CCS2	27	0.1	0.13	0.36	0.53	0.68	0.76	0.85	0.90	0.94
CCS3	12	0	0.48	0.77	0.90	0.94	0.96	0.99	0.99	0.99
CCS4	11	0.03	0.63	0.91	0.97	0.97	0.98	0.98	0.99	0.99
NIST1	63	0	0	0.01	0.03	0.13	0.24	0.33	0.41	0.50
NIST2	11	0	0.07	0.63	0.84	0.96	0.99	0.99	0.99	0.99

Table 3.1: Minimum values of b to achieve column recovery rate equal to 1, and heuristic column recovery when b is fixed to the listed values.

each parameter set and noise level $\sigma = 0.0096$ (the most aggressive of my scale), I ran around 10^3 experiments and looked at the smallest beam b for which all experiments ended with the actual secret $\mathbf{s}$ as part of the final beam (allowing an adversary to identify $\mathbf{s}$ by a subsequent enumeration of all final beam candidates). The resulting values are reported in the $b_{\min}$ column of Table 3.1. With the exception of CCS1, I notice that $b_{\min}$ is at most 2^6, so again only a few bits of security remain. As $b_{\min}$ will invariably grow as the number of experiments does, until eventually it is as large as the key space, in the second set of experiment I estimated final vector recovery rate as a function of the beam size, for $b \leq 10$. The results are again reported in Table 3.1 and are fairly damning: even for NIST1 a recovery rate of around 50% is achieved.

It is very interesting to notice how in Table 3.1, CSS1 seems to be particularly tough to beat. As I will describe in detail in the next and last part of this chapter, this has to do with how few subtraces are available there.

3.5 Choosing your parameters

So far, I have compared increasingly effective attack strategies, where I compared different parameter sets purely by name, so without further reference to their actual parameters. I now investigate the effect of these parameters on the efficacy and efficiency of the attack. Specifically, I consider the effects of n and $\left|\mathrm{Supp}\left(\chi\right)\right|$ on the natural side-channel vulnerability of the resulting matrix–vector multiplication.

Figure 3.7 provides a scatter plot of $(n, \left|\mathrm{Supp}\left(\chi\right)\right|)$ for the various parameter sets suggested [BCD+16, NAB+17]. Furthermore, I encoded the success rate of the extend-and-prune attack with beam $b = 1$ (Section 3.4.1) and realistic noise level (dashed line

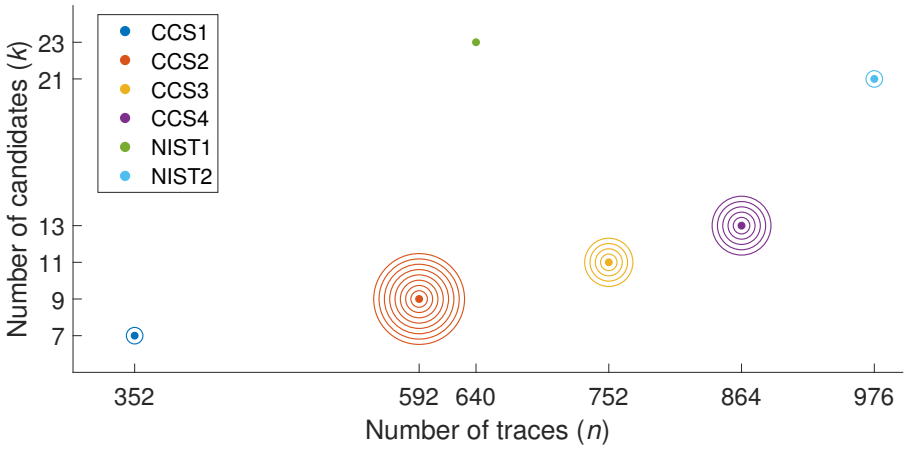


Figure 3.7: Visual representation of all parameter sets. For each of them, the x axis lists n, and the y axis lists $\left|\mathrm{Supp}\left(\chi\right)\right|$. The number of concentric circles around each parameter set encodes how successful my attack is against it.

in Figure 3.6b) with concentric circles around each parameter set. The number of circles is simply the ceil of said success rate times ten, and is helpful in visually quantifying the outcome I achieved in each setting.

The effect that the choice $(n, \left|\mathrm{Supp}\left(\chi\right)\right|)$ has on the hardness of the LWE instance has been well studied [APS15], but from a side-channel perspective, new meaning emerges: n corresponds both to the number of (sub)traces an adversary obtains on each component of $\mathbf{s}$ *and* to the number of positions to retrieve, whereas $\left|\mathrm{Supp}\left(\chi\right)\right|$ quantifies the keyspace size for individual positions.

Although the divide-and-conquer attack suffers badly when more positions need to be recovered, the extend-and-prune approach is far more robust in this respect. For instance, the main difference between CCS1 and CCS2 is that n is two times larger in CCS2 than in CCS1, thus providing a much easier target for my attack. Thus increasing n overwhelmingly has the effect of making life easier for an adversary as more leakage will be available. In other words, while increasing the dimension n makes the LWE instance *harder*, it makes the underlying matrix–vector multiplication *easier* to attack in the side-channel scenario. This conclusion does rely on square $\mathbf{A}$, so $n = m$. In case $\mathbf{A}$ is a non-square matrix, then m refers to the number of traces and n to the number of positions to recover. The hardness of LWE appears to be mainly governed by n, where increasing n makes both the LWE instance harder and it complicates side-channel cryptanalysis. Similarly, both for LWE and for the side-channel analysis, increasing m makes attacks potentially easier, with the effect for side-channels much more pronounced.

The qualitative effect of increasing $\left|\mathrm{Supp}\left(\chi\right)\right|$ is straightforward: a large keyspace means that there are more options to choose from, with corresponding signals that are closer together, making distinguishing harder. This effect is illustrated by comparing the two parameter sets NIST1 and CCS2. These two sets have roughly equal n, but NIST1's $\left|\mathrm{Supp}\left(\chi\right)\right|$ is about three times that of CCS2: my attacks confirm that CCS2 is a lot easier to attack than NIST1.

3.5.1 Effect of modifying NIST1

I conducted a final experiment to gain more insights on parameter set selection. I focused my attention on the two NIST parameter sets: they have roughly the same k (it differs by only two) but NIST1 has two thirds fewer traces than NIST2. I therefore increased n in NIST1 to match NIST2's ($n = 976$) and analysed the extend-and-prune attack in two settings: when $b = 1$ and σ is the realistic value, and when $b = 10$ and $\sigma = 0.0096$, i.e. the worst noise level I have considered so far. In the former case the success rate increased from 0.01 to 0.11, almost equating the success rate of 0.12 observed in the NIST2 setting. In the $b = 10$ case, the success rate reported in Table 3.1 (0.50) skyrocketed to 0.94, again very close to NIST2's. This strongly indicates how having larger matrices, hence more traces per secret element, goes in favour of the adversary. Therefore in general being overpessimistic in the choice of n might prove fatal if side-channel attacks are a concern.

Chapter 4

Swapping sides: from offence to defence

The structure of Frodo, and in general the kind of operations involved in LWE-based schemes, are rather peculiar. As shown in Chapter 3, the iterative structure of matrix multiplications can be exploited to mount the fairly devastating extend-and-prune attack, along with the fact that secret elements take values from an unusually small set of possibilities compared to pre-quantum schemes.

In the present chapter, I start off from where I left: extend-and-prune. I will show two strategies to protect against it, with different trades-off. Then, to counteract divide-and-conquer, I first explore how the structure of Frodo can be leveraged to build up defences, and finally show how well-known countermeasures from the literature can be integrated in the protocol.

The content of this chapter, with the exception of Section 4.1 which was part of the original paper [BFM+18a], is mostly unpublished. Nonetheless, I am the main author and writer. In particular, Section 4.2 is based on an original idea, which I developed specifically for this thesis, while I claim no paternity on the ideas contained in Section 4.3, which are established facts from the literature. My contribution in the latter section has been to apply them in the context of Frodo.

Analysis of countermeasures can follow two orthogonal directions. First of all, it is desirable that the overhead introduced to protect an implementation is low. I address the performance of the countermeasures I propose here in Chapter 5. The other direction is their effectiveness in the setting they are designed to protect. This can be assessed with experiments or with an informed analysis on their structure: I use both such options in my analysis. Sections 4.1 and 4.3 contain an informed analysis, while I

develop some experiments in Section 4.2.

4.1 Thwarting extend-and-prune

As seen in Chapter 3, extend-and-prune is a particularly devastating template attack: it needs a single trace only and its success rate is rather high, especially when the pruning strategy retains more than one candidate per round. On the downside, extend-and-prune is significantly more difficult to mount than a divide-and-conquer using templates, as it requires on-the-fly templating.

Extend-and-prune leverages incremental computations over a partial result to build templates, including a wider time window than those generated when performing divide-and-conquer. Three key elements that the extend-and-prune technique exploits are:

1. the initial state of the system is known, this includes the first value of the accumulator (which is zero) and values of internal registers holding elements of $\mathbf{A}$ and $\mathbf{S}$;

2. previously recovered secret elements are leveraged to reduce algorithmic noise and to formulate an educated guess on the value of the accumulator;

3. the same incremental computations over the partial result, again the accumulator, being executed in more than one portion of the trace. This is due to the single-trace context.

What makes point 2 possible is that, in the implementation I considered in Section 3.2, the accumulator is iteratively summed with ring multiplications. Doing so exposes intermediate partial sums, since adaptive templates are used. Point 3, instead, simply follows from how textbook matrix multiplication is implemented.

Initialising the accumulator with a non-zero unknown value seemingly addresses point 1. There are several options to achieve this, which I will discuss more in detail in Section 4.1.1.

Points 2 and 3 are more structural, and effectively signify that the computation is performed in the same way for every row of $\mathbf{A}$. In other words, the first position of the secret is always the first one to be multiplied by an element of $\mathbf{A}$. This seeming alignment holds every time a column of $\mathbf{S}$ is involved, i.e. n times. Thus, the adversary exploits multiple trace chunks for the whole computation, using previously recovered secrets to

reduce algorithmic noise and therefore enhance leakage on the currently targeted one. Section 4.1.2 shows how to break such a pattern.

4.1.1 Masking the accumulator

The most natural initial assignment of the accumulator is hinted by Algorithm 11 itself: $\mathbf{b} \leftarrow \mathbf{e}$ on Line 1. That is to say, instead of starting from 0, the accumulator is set to the correct entry of $\mathbf{E}$. Despite the simplicity, the security gain is marginal: since the error follows the same narrow distribution of the secret, the amount of effort on the adversary's side to overcome such a patch is simply given by the generation of $\left|\mathrm{Supp}\left(\chi\right)\right|^2$ templates for the first iteration. This is equivalent to mounting an extend-and-prune attack on a vector of length $n+1$ with a pruning strategy that retains all candidates for the first position. In other words, the formulation is equivalent to attacking

$$\begin{pmatrix} 1 & A_{r,1} & \dots & A_{r,n} \end{pmatrix} \begin{pmatrix} e_r \\ s_1 \\ \vdots \\ s_n \end{pmatrix}$$

where the adversary is required to build templates for the first two positions of the newly built secret, as no actual operation involves e_r.

A more effective alternative is to introduce randomness in the process and actually mask the accumulator. The latter is therefore initialised to a random element of $\mathbb{Z}_q$, ring multiplications are added in and, at the very end, the initial value is subtracted from the sum. Algorithm 12 implements such an approach, where red lines highlight differences compared to Algorithm 11. In particular, Lines 3 and 4 are where a random number is sampled and used to initialise the accumulator sum. The mask is later removed on Line 7.

Templates would now need to include guesses for the initial value of the accumulator, as well as guesses for the first position of the secret. Differently than before, where the error term did not significantly increase the effort, this randomised approach requires the adversary to build $q\,\mathrm{Supp}\left(\chi\right)$ templates. Furthermore, a very large number of them should be retained as part of the pruning strategy, making the whole attack significantly more cumbersome.

Masking the accumulator effectively makes its exploitation infeasible, as a similar argument as in Section 3.3.4 applies: an adversary would see the previous partial sum

Algorithm 12 Matrix–vector multiplication with masked accumulator.

Input: $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$; $\mathbf{s}, \mathbf{e} \in \mathbb{Z}_q^n$
Output: $\mathbf{b} \leftarrow \mathbf{As} + \mathbf{e}$

1: $\mathbf{b} \leftarrow \mathbf{e}$
2: **for** $r = 1, \ldots, n$ **do**
3: $\quad x \xleftarrow{\$} \mathbb{Z}_q$
4: $\quad sum \leftarrow x$
5: $\quad$ **for** $i = 1, \ldots, n$ **do**
6: $\quad\quad sum \leftarrow sum + A_{r,i} \cdot s_i$
7: $\quad b_r \leftarrow (b_r + sum - x) \bmod q$
8: **return b**

as a uniformly distributed random variable. Differently than divide-and-conquer, however, the order of operations can still be exploited: previously retrieved secret elements can be used to decrease algorithmic variance when performing the `ldrh` and `muls` instructions. On top of that, masking the accumulator requires randomness, which might be problematic to generate on some devices. For these reasons, I offer an alternative countermeasure to extend-and-prune.

4.1.2 Shifting rows

As previously mentioned, every row of $\mathbf{A}$ is multiplied by the target column of $\mathbf{S}$ in the same way. In other words, s_1 is always the first element used, s_2 always the second and so on. This regularity is exploited by the adversary in that she has n trace chunks for s_1 where the accumulator is known (to be 0), thus making the latter's retrieval easier. Guesses for s_1 are leveraged to predict the accumulator and obtain an advantage when targeting s_2, up until s_n. Breaking the pattern is therefore enough to thwart extend-and-prune.

Simply shuffling the order in which elements from $\mathbf{A}$ and $\mathbf{s}$ are multiplied together suffices for the purpose. Specifically, if s_1 was the first element to be multiplied when the first row of $\mathbf{A}$ is processed, the second when the second row of $\mathbf{A}$ is used and so on, there would not be a single trace chunk where s_1 is preceded by the same computation, hence making the exploitation of any "previous" knowledge infeasible.

Algorithm 13 is yet again the matrix–vector multiplication, with the red lines being the difference compared with the unprotected version. In particular, on Line 6 I compute the same summation as before, but with a different order of addends each time. In

Algorithm 13 Matrix–vector multiplication with rows of **A** shifted.

Input: $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$; $\mathbf{s}, \mathbf{e} \in \mathbb{Z}_q^n$
Output: $\mathbf{b} \leftarrow \mathbf{As} + \mathbf{e}$

1: $\mathbf{b} \leftarrow \mathbf{e}$
2: **for** $r = 1, \ldots, n$ **do**
3: $\quad sum \leftarrow 0$
4: $\quad$ **for** $i = 1, \ldots, n$ **do**
5: $\quad\quad j \leftarrow (i + r - 2) \bmod n + 1$
6: $\quad\quad sum \leftarrow sum + A_{r,j} \cdot s_j$
7: $\quad b_r \leftarrow (b_r + sum) \bmod q$
8: **return b**

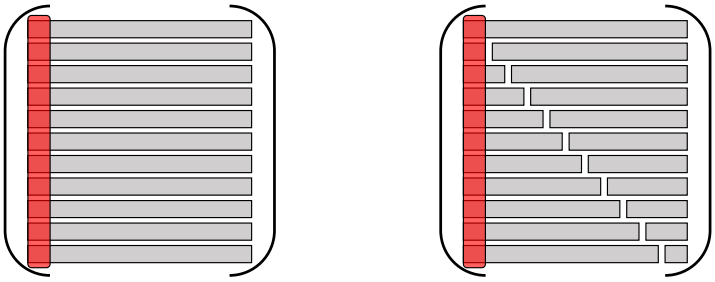


Figure 4.1: Visualisation of the shift countermeasures. Rows are indicated by grey boxes, while the red one highlights elements multiplied as first ones in a matrix multiplication. The left matrix is **A** when unprotected, while the right one is **A** after the shift is applied.

formulae, **b** is computed as

$$
\begin{aligned}
b_1 &= A_{1,1} \cdot s_1 + A_{1,2} \cdot s_2 + \ldots + A_{1,n-1} \cdot s_{n-1} + A_{1,n} \cdot s_n \\
b_2 &= A_{2,n} \cdot s_n + A_{2,1} \cdot s_1 + \ldots + A_{2,n-2} \cdot s_{n-2} + A_{2,n-1} \cdot s_{n-1} \\
&\vdots \\
b_n &= A_{n,2} \cdot s_2 + A_{n,3} \cdot s_3 + \ldots + A_{n,n} \cdot s_n + A_{n,1} \cdot s_1
\end{aligned}
$$

where operands are added exactly as written. Figure 4.1 is a graphical representation of the same process: row i, drawn as a grey box, is shifted $i - 1$ positions to the right. The vertical red box highlights the secret elements which are multiplied first in both cases: unprotected (left) and protected (right).

The two key ideas that break extend-and-prune are as follows.

- Knowledge of the accumulator being zero helps only for a single trace chunk per secret position, i.e. when computing $A_{i,i} \cdot s_i$ for all i, which is far from the statistical

significance needed to mount any attack. Similarly to the conclusions drawn for masking the accumulator, the latter is hardly exploitable.

- Retrieving "early" secrets and leveraging them to attack "later" ones is infeasible. This is due to the fact that s_0 has exactly the same attack surface as all other positions in terms of information known by an adversary. This is in contrast to masking the accumulator, which left the exploitation of previous secrets open.

Theoretically speaking, extend-and-prune works better than divide-and-conquer even if the shifting is applied, however to successfully mount it one would need to almost never prune the guesses tree. Clearly, having such a computational power would also open the way to brute force, making all attempts at using extend-and-prune of little practical interest.

4.1.3 Summary

Sections 4.1.1 and 4.1.2 presented two potential countermeasures to thwart the extend-and-prune strategy. Masking the accumulator is the most obvious and natural solution, although it has two major drawbacks: requiring randomness and allowing for exploitation of previously recovered secrets. On the contrary, changing the order of addends while performing matrix–vector multiplication is completely deterministic, yet structurally prevents many of the benefits extend-and-prune has over divide-and-conquer. Although theoretically still possible, extend-and-prune against shifting would be infeasible to mount.

The main reason why they are both of interest lies in their performances. As I will show in Chapter 5, shifting is really challenging to implement if the matrix $\mathbf{A}$ is not fully present in memory, as is the case for devices in the IoT ecosystem. There, randomness is also not a severe limitation, rendering masking the accumulator preferable over shifting.

With the Frodo protocol switching from KEP to KEM, the multi-trace setting is ever more concerning. On the one hand, the attack surface increases as techniques from that setting, e.g. DPA, apply. On the other hand, classic countermeasures can also be used to protect an implementation.

4.2 Hamming weight model

As briefly explained in Section 2.3, the power consumption of a device is modelled by the sum of three components: the exploitable part, the switching noise (or algorithmic variance) and the environmental noise. Leakage is found in the first component, and is itself the object of a mathematical model. In other words, leakage is a function of the processed data, whose mathematical representation is often unknown. To cope with this uncertainty, two approaches are possible: formulate a working assumption about the function leakage follows, or learn it from the device.

I have already explored the second approach in Chapter 3, that is to say templating the target device. The first approach, instead, entails to pick a reasonable mathematical function and to work under the assumption that leakage behaves apparently according to it. How reasonable a choice can be, mainly depends on how strong the relation between the chosen function and the underlying hardware is. For example, buses might leak the Hamming weight of the carried bits, while registers might leak the Hamming distance of inputs when they are updated, since these functions are closely related to how power is used by these components [MOP07, Section 3.3].

4.2.1 Homogenising Hamming weight

ELMO's traces are a simulation of how triplets of assembly instructions and their operands contribute toward the power consumption of an ARM Cortex-M0 [MOW17]. Because they are based on real experiments, they essentially fall into the second category mentioned above: they represent the outcome of templating that particular microprocessor.

In the present subsection, instead, I assume leakage is shaped as the Hamming weight of internal quantities. Despite the fact that building a power model is usually done by adversaries prior to mount attacks, as having a well defined and simple mathematical model greatly simplifies computations, I present how the peculiar structure of Frodo allows for a simple countermeasure that decreases signal to noise ratio. The idea is to transform the secret such that all elements have the same Hamming weight. This way, an adversary would no longer be able to discern guesses based on their predicted Hamming weight, making finding the right secret more challenging.

As it always happens when reasoning around power models, estimating their practicality, i.e. assessing whether the assumptions they are built upon find practical instantiation in the real-world, is almost solely based on cases. This is due to the fact that

Index	$2h-1$	…	$h+k+1$	$h+k$	$h+k-1$	…	h	$h-1$	…	$k+1$	k	$k-1$	…	0
$\langle a \rangle$	$\langle x \rangle_{h-1}$	…	$\langle x \rangle_{k+1}$	1	0	…	0	0	…	0	0	0	…	0
$\langle x \rangle$	0	…	0	0	0	…	0	$\langle x \rangle_{h-1}$	…	$\langle x \rangle_{k+1}$	1	0	…	0

Table 4.1: Binary representation of the two operands a and x.

is very difficult to find two distinct devices that share the leakage behaviour, let alone to find a power model that correctly represent a large class of them. In other words, a power model that fits well for a setup, might fail spectacularly on another. This was the case, for instance, for the divide-and-conquer attach in Section 3.3: Aysu et al. [ATT+18] comfortably worked with a Hamming distance power model, which did not work for me. Devices that do leak Hamming weight of some intermediates, depending on where and when the power measurement is conducted during computations, exist and have been attacked in the past, see for example [MOP07, Figure 4.5]. This means that assuming an adversary has access to measurements which can be meaningfully analysed with Hammin weight of some intermediate values is sound.

Lemma 4.1. *Let $h > 1$ be an integer. Then for all integers $0 < x < 2^h$, it holds that*

$$H\left(x \cdot \left(2^h - 1\right)\right) = h$$

where $H(\cdot)$ denotes Hamming weight.

Proof. Since $x < 2^h$, $\langle x \rangle$, which is the binary representation of x, is at most h bits long. Moreover, since $x \neq 0$, it has at least a 1. Let $H(x) = t > 0$ and let $0 \leq k < h$ be the smallest index such that $\langle x \rangle_k = 1$. For notational convenience, let $a = x \cdot 2^h$ and let

$$b = x \cdot \left(2^h - 1\right) = a - x \ .$$

Table 4.1 lists all bits in $\langle a \rangle$ and $\langle x \rangle$, together with their indexes.

The goal is to count how many 1s $\langle b \rangle$ has. First of all, $\langle b \rangle_0, \ldots, \langle b \rangle_{k-1}$ are equal to zero, because so are the corresponding bits of a and x. Then $\langle a \rangle_k = 0$ but $\langle x \rangle_k = 1$, which means a 1 needs to be carried from the first position of $\langle a \rangle$ being equal to one. By construction, that is $\langle a \rangle_{h+k}$ because the first h positions of $\langle a \rangle$ are zeros and so are the next k, being them equal to the first k positions of $\langle x \rangle$. Moving the carry and subtracting the kth bit yields $\langle b \rangle_k = 1$, $\langle a \rangle_{h+k} = 0$ and $\langle a \rangle_{h+k-1} = \ldots = \langle a \rangle_{k+1} = 1$.

This is summarised in Table 4.2, where $\langle a \rangle$ is shown after the $(h+k)$th bit has been carried to position k. The binary representation of x is reported again, so that bit-wise

Index	$2h-1$	...	$h+k+1$	$h+k$	$h+k-1$	...	h	$h-1$	...	$k+1$	k	$k-1$	...	0
$\langle a \rangle$	$\langle x \rangle_{h-1}$	...	$\langle x \rangle_{k+1}$	0	1	...	1	1	...	1	10	0	...	0
$\langle x \rangle$	0	...	0	0	0	...	0	$\langle x \rangle_{h-1}$	...	$\langle x \rangle_{k+1}$	1	0	...	0
$\langle b \rangle$	$\langle x \rangle_{h-1}$	...	$\langle x \rangle_{k+1}$	0	1	...	1	$\overline{\langle x \rangle_{h-1}}$	...	$\overline{\langle x \rangle_{k+1}}$	1	0	...	0

Table 4.2: Binary representation of a once $\langle a \rangle_{h+k}$ has been carried to position k, of x and of their subtraction b. Note that in column k, 10 is the binary representation of 2.

subtraction can now be computed for all the remaining bits of $\langle b \rangle$. Having made the latter explicit, simply counting the ones concludes the proof:

$$\begin{aligned} H(b) &= \sum_{i=0}^{2h-1} \langle b \rangle_i \\ &= \sum_{i=0}^{k-1} \langle b \rangle_i + \langle b \rangle_k + \sum_{i=k+1}^{h-1} \langle b \rangle_i + \sum_{i=h}^{h+k-1} \langle b \rangle_i + \langle b \rangle_{h+k} + \sum_{i=h+k+1}^{2h-1} \langle b \rangle_i \\ &= 0 + 1 + \sum_{i=k+1}^{h-1} \overline{\langle x \rangle_i} + k + 0 + \sum_{i=k+1}^{h-1} \langle x \rangle_i \\ &= 1 + (h - (k+1) - (t-1)) + k + t - 1 \\ &= h \ . \end{aligned}$$

□

Lemma 4.1 gives a constructive way of homogenising the Hamming weight of integers less than a power of two. In order to apply it to my setting, however, a few caveats need to be addressed.

4.2.2 Application to a secret matrix

The overall idea is to apply a function f to $\mathbf{S}$ component-wise, such that $f(\mathbf{S})$ is a matrix whose elements all have the same Hamming weight. Clearly, f should be derived from Lemma 4.1, thus its hypotheses must hold. The first problem is that $\mathbf{S}$ contains many zeros, which is in fact the most common value. Since $\mathrm{Supp}\left(\chi\right) = \{-s, \ldots, s\}$ modulo q, for a certain integer s, a simple modular shift by $s+1$ suffices to map all its elements to non-zero values.

A second caveat is that Lemma 4.1 is stated for integers, and is actually not valid if $x \cdot \left(2^h - 1\right)$ wraps around modulo q. All that is needed to take care of such an eventuality

Parameter set	s	q	Admissible h
CCS1	3	2^{11}	3,4,5,6,7,8
CCS2	4	2^{12}	4,5,6,7,8
CCS3	5	2^{15}	4,5,6,7,8,9,10,11
CCS4	6	2^{15}	4,5,6,7,8,9,10,11
NIST1	11	2^{15}	5,6,7,8,9,10
NIST2	10	2^{16}	5,6,7,8,9,10,11

Table 4.3: Admissible h for each parameter set, i.e. for which the hypotheses of Lemma 4.1 hold.

is to check whether there exists a h such that, for all $z \in \{-s, \dots, s\}$,

$$\begin{cases} (z+s+1)\cdot(2^h-1) < q \\ 0 < z+s+1 < 2^h \end{cases}$$

where both equations must be true in $\mathbb{Z}$. Numbers are small enough so that a simple exhaustive search finds the admissible values for h. Table 4.3 lists them all for each parameter set.

Since the h listed in Table 4.3 are such that the hypotheses of Lemma 4.1 are met, they also correspond to the Hamming weight after the function

$$f(z) = (z + (s+1)) \cdot \left(2^h - 1\right) \pmod{q}$$

is applied, for all $z \in \text{Supp}(\chi)$. Seen as an operation among matrices, the latter becomes

$$\tilde{\mathbf{S}} = \left(2^h - 1\right)(\mathbf{S} + (s+1)) \pmod{q}$$

where $\tilde{\mathbf{S}}$ is a matrix whose positions have Hamming weight h. Note that from a computational point of view, the latter can be implemented with one addition, one subtraction and one shift, hence avoiding multiplications.

4.2.3 Integration to Frodo

Changes to $\mathbf{S}$ must be compensated somewhere else in the protocol to preserve correctness. The decision on where to apply and where to remove a countermeasure is tightly coupled to the attacker model: it is important to be clear about what the countermeasure is protecting against, which necessarily means being clear about what may attack the system. This was the case for the countermeasure described in Section 4.1, which took care of the extend-and-prune strategy.

The Hamming weight homogenisation aims at thwarting the divide-and-conquer strategy when the defender knows leakage follows the Hamming weight model and can be applied both in the single-trace and multi-trace settings.

In general, every time there is an homogenised secret matrix $\tilde{\mathbf{S}}$ which is multiplied by a matrix $\mathbf{M}$, as

$$\mathbf{M}\tilde{\mathbf{S}} = \mathbf{M}\left(\left(2^h - 1\right)(\mathbf{S} + (s+1))\right) = \left(2^h - 1\right)(\mathbf{MS} + \mathbf{M}((s+1)\mathbf{1}_{n\times\overline{n}})) \pmod{q}$$

where $\mathbf{1}_{n\times\overline{n}}$ is a matrix with the specified dimensions whose elements are all equal to one, the correction factors to obtain the correct results are

$$\mathbf{MS} = \left(2^h - 1\right)^{-1}\mathbf{M}\tilde{\mathbf{S}} - \mathbf{M}((s+1)\mathbf{1}_{n\times\overline{n}}) \,.$$

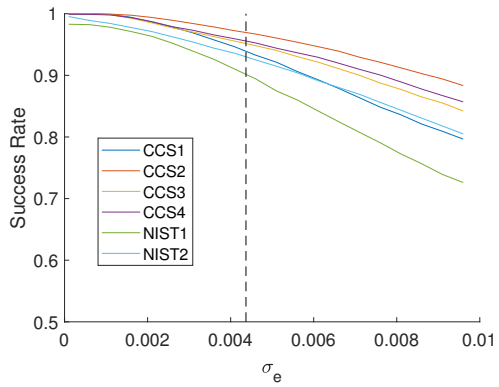
The first multiplicative factor exists because $2^h - 1$ always has an inverse modulo a power of two, since they are coprime. The latter and the subsequent additive factor are solely based on public knowledge, hence not sensitive. A very similar line of reasoning holds if $\tilde{\mathbf{S}}$ is multiplied to the right.

This is directly applicable in a single-trace setting, as every computation involving a secret matrix is potentially a target, even the ephemeral ones, e.g. $\mathbf{S}'$. Note that the transformation itself does not depend on which value a position of $\mathbf{S}$ takes, and that also the correction factors are independent of the secret.

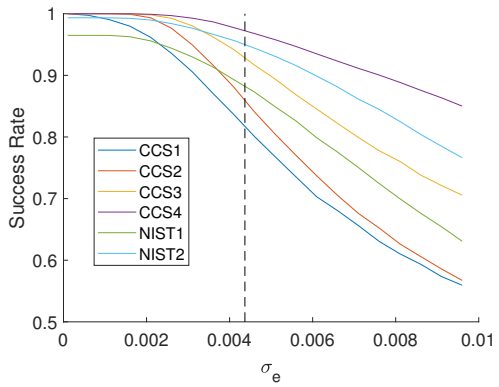
4.2.4 Experimental results

As I mentioned at the beginning of the section, ELMO traces are produced based on the contribution of several quantities: multiplicative factors based on the triplet of instructions considered, Hamming distance between some operands, Hamming weight of operands, etcetera. On the one hand, therefore, the Hamming weight of secret elements certainly are part of the computational model producing traces, on the other many more quantities are factored in. In other words, my working assumption that leakage is exclusively a function of Hamming weight of the input does not hold for ELMO traces, but is nonetheless interesting to assess the effect of Hamming weight homogenisation on ELMO traces.

Figure 4.2 compares the success rate of divide-and-conquer with the improved distinguisher, see Section 3.3.3, with the same attack but where Hamming weight homogenisation was applied. In particular, note that Figure 4.2a is the same as Figure 3.4b and is replicated here to simplify visual comparison with Figure 4.2b, which instead reports success rate against Hamming weight homogenisation. Both figures target two



(a) Success rate of divide-and-conquer with improved distinguisher. Same as Figure 3.4b.



(b) Success rate of divide-and-conquer with improved distinguisher against an homogenised secret matrix.

Figure 4.2: Comparison between success rates of divide-and-conquer with and without Hamming weight homogenisation, against a single position.

points of interest (loading of the secret and multiplication) and refer to position success rate.

Overall the success rate degrades, meaning that Hamming weight homogenisation does indeed reduce the amount of signal an adversary can exploit even in a context where the initial working assumption does not hold entirely. Clearly, Figure 4.2b has been obtained by including the modification to the scheme also in the templates: since the countermeasure is deterministic and contains no secret transformation, an adversary can, and has to, account for it in the templating phase. Moreover, Hamming weight of the secret is certainly not the only metric related to secret values that can leak sensitive information, which is why, despite the degraded performance, Figure 4.2 shows two very similar behaviours.

What is even more interesting to notice, however, is the exceptional cases to the above discussion. There are situations, in the divide-and-conquer attack, where secret elements and their associated quantities do not fall under the signal umbrella, but under the switching noise (or algorithmic variance) one. In such cases, reducing the contribution of Hamming weight could even *improve* the attack: this seems to be the case for the CCS4 parameter set and for the NIST2 parameter set in the low-noise regime.

All in all, Hamming weight homogenisation is an interesting countermeasure in theory, as it exploits a peculiar algebraic structure of small secrets in LWE. Moreover, it shows that ideas usually applied to simplify attacks, i.e. assuming a known and well-defined leakage profile rather than templating the device, can turn useful in defending

cryptoschemes too. In practice, however, this is the perfect example of how tricky it is to affect the signal-to-noise ratio by tuning only one or few quantities involved, and how doing so can backfire in certain cases. This discussion calls for more structural approaches, which are coming up next.

4.3 Other countermeasures from the literature

Sections 4.1 and 4.2 described alternative implementations of some algorithms in the Frodo protocol, mostly applicable to both KEM and KEP versions, to protect against the two attacks I showed in Chapter 3. In this case, "protecting" takes different meaning depending on the attack and on the countermeasure.

- The extend-and-prune attack strategy from Section 3.4 was experimentally proven to be the best among the proposed combinations of strategies and distinguishers. It was in a sense optimal, because it exploited all available leakage from the three points of interest making up a $\mathbb{Z}_q$ multiplication in my assembly implementation. Luckily, the assumptions on which it is based are also easy to falsify through a fairly straightforward alternative implementation, as shown in Section 4.1. In this case, therefore, the protection mostly thwarts the attack.
- The divide-and-conquer strategy from Section 3.3 is weaker but easier to apply and faster to run. Moreover, the countermeasure I presented in Section 4.2 does not structurally prevent the attack, but merely contributes to render it harder only in the situation where leakage can be modelled with Hamming weight. Despite the fact that it is interesting how Frodo can benefit from an assumption that is usually adopted to attack schemes, how to build a more solid defence against divide-and-conquer remains an open point.

For these reasons, I now focus on other ways to prevent divide-and-conquer attacks in the multi-trace setting. Indeed, Sections 4.1 and 4.2 were about thwarting extend-and-prune and lowering the signal to noise ratio in devices leaking Hamming weight, respectively. Single-trace scenarios indeed suffer from a low signal to noise ratio as the total number of power traces available is very limited. This is not the case for multi-trace settings where, if the ratio is low, an adversary could in principle simply acquire more traces.

I will analyse two widely known countermeasures from the literature and apply them to the FrodoKEM protocol. Most of the content of the current section is, there-

Algorithm 14 FrodoKEM.DECAPS with blinding

Input: Ciphertext $c = (\mathbf{B}', \mathbf{C}, \mathbf{d}) \in \mathbb{Z}_q^{\overline{m} \times n} \times \mathbb{Z}_q^{\overline{m} \times \overline{n}} \times \{0,1\}^{\text{LEN}}$, secret key $sk = (\mathbf{s}, \mathbf{S}) \in \{0,1\}^{\text{LEN}} \times \mathbb{Z}_q^{n \times \overline{n}}$ and public key $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B}) \in \{0,1\}^{128} \times \mathbb{Z}_q^{n \times \overline{n}}$.
Output: Shared secret $\mathbf{ss} \in \{0,1\}^{\text{LEN}}$.

1: $\text{seed}_{\mathbf{X}} \xleftarrow{\$} \mathcal{U}(\{0,1\})^{128}$
2: $\mathbf{X} \leftarrow \text{cSHAKE}(\text{seed}_{\mathbf{X}}, 16n\overline{m}, 0)$
3: $\mathbf{B}'_{\mathbf{x}} \leftarrow \mathbf{B}' - \mathbf{X} \pmod q$
4: $\mathbf{M}_{\mathbf{x}} \leftarrow \mathbf{C} - \mathbf{B}'_{\mathbf{x}}\mathbf{S} \pmod q$
5: $\mathbf{M} \leftarrow \mathbf{M}_{\mathbf{x}} - \mathbf{X}\mathbf{S} \pmod q$
6: $\mu' \leftarrow$ FrodoKEM.Decode($\mathbf{M}$)
7: **return** FrodoKEM.CCA(c, $\mathbf{s}$, μ', pk)

fore, public knowledge and inspired by techniques already present in the literature. For benchmarking purposes, Chapter 5 will implement some of the ideas explained here.

4.3.1 Blinding

Side-channel adversaries make use of public information, e.g. the matrix $\mathbf{B}'$ during DECAPS, and assumptions on leakage to produce guesses which are then compared against actual (or, in my case, simulated) power measurements. One way to prevent attacks, therefore, is to take public information out of the equation by *blinding* it. In other words the input, being it the plaintext in symmetrical ciphers or the ciphertext in public key ciphers, is hidden with some internally generated random value. Depending on the scheme, this can mean either adding or multiplying a random value to the input, as is the case of RSA [MOP07, Section 9.1.3] where the plaintext is summed with a random value to avoid secret exponent recovery, or exploiting the homomorphic nature of a scheme to add the encryption of a random message to a ciphertext. This latter approach has been followed by Reparaz et al. [RdCR+16] in the context of Ring-LWE.

In the case of FrodoKEM, the only multiplication to secure is that between $\mathbf{B}'$ and $\mathbf{S}$ during DECAPS, as it involves the long-term secret matrix and is performed multiple times. First of all, 128 bits of fresh randomness are generated, which are then interpreted as a seed and expanded into an $\overline{m} \times n$ matrix. Such a matrix is then added to the ciphertext $\mathbf{B}'$, the result is multiplied by $\mathbf{S}$ and, finally, a correction factor is applied to restore correctness.

Algorithm 14 shows the modifications needed to blind $\mathbf{B}'$ inside the DECAPS algorithm. In particular, a seed is first produced because it is much shorter than the ex-

panded matrix. At first glance it might seem that the situation has worsened since now two multiplications involving $\mathbf{S}$ exist. However, both of them have unknown operands. Since $\mathbf{X}$ is generated afresh and on-the-fly, an adversary does not know it, preventing any formulation of guess for the $\mathbf{XS}$ multiplication. Moreover, since $\mathbf{X}$ is uniformly random in $\mathbb{Z}_q$, so is $\mathbf{B}' + \mathbf{X}$ (it is effectively one-time-padding it). Therefore guesses cannot be formulated there either.

Ciphertext blinding also helps in other attack scenarios, for instance against adversaries who can tamper with the ciphertext to manipulate how the subsequent multiplication by $\mathbf{S}$ is carried out. The main drawback, however, is that the secret is still intact and fully present on the device, opening the way for other potential attack vectors.

4.3.2 Masking

Some of the concepts above can be applied directly to the secret, in such a way that the actual protection is implemented directly upon the sensitive material, rather than blinding the operations involving it. In this case, the technique is called *masking* [MOP07, Section 9.1] and it should come as no surprise that the book by Mangard, Oswald and Popp describes blinding and masking together: the underlying idea is very similar, since it boils down to splitting the secret in two: a mask which is the equivalent of $\mathbf{X}$ and a masked value which is equivalent to $\mathbf{B}_\mathbf{x}$ in Algorithm 14.

Differently than ciphertext blinding, since $\mathbf{S}$ is a long-term matrix, masking can be performed during KEYGEN in such a way that the mask and masked value, also called *shares* of the secret matrix, are kept separate at all times. Clearly, this comes at the cost of almost doubling the size of the secret key, since two matrices must be stored.

Algorithms 15 and 16 show the masking scheme applied to FrodoKEM. As I mentioned, the secret matrix $\mathbf{S}$ is split in two shares during KEYGEN and, while the public key is computed with the full matrix, the two shares are returned separately as part of the secret key. They are used in a similar way as Algorithm 14 during DECAPS to compute the matrix $\mathbf{M}$, however an extra step is needed. Shares must be refreshed, i.e. a random matrix must be added to $\dot{\mathbf{S}}$ and subtracted by $\ddot{\mathbf{S}}$ to preserve correctness, otherwise an adversary would be able to collect multiple traces about both immutable shares and the divide-and-conquer attack would still be feasible. Refreshing shares does not hinder their main properties, that is to say that $\dot{\mathbf{S}} + \ddot{\mathbf{S}} = \mathbf{S} \pmod q$.

Algorithm 15 FrodoKEM.KeyGen with masking

Input: None.

Output: Key pair $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B}) \in \{0,1\}^{128} \times \mathbb{Z}_q^{n \times \overline{n}}$ and $sk = (\mathbf{s}, \dot{\mathbf{S}}, \ddot{\mathbf{S}}) \in \{0,1\}^{\text{LEN}} \times \mathbb{Z}_q^{n \times \overline{n}} \times \mathbb{Z}_q^{n \times \overline{n}}$.

1: $\mathbf{s} \| \text{seed}_{\mathbf{E}} \| \mathbf{z} \xleftarrow{\$} \mathcal{U}(\{0,1\})^{3\text{LEN}}$
2: $\text{seed}_{\mathbf{A}} \leftarrow \text{cSHAKE}(\mathbf{z}, 128, 0)$
3: $\mathbf{A} \leftarrow \text{FrodoKEM.Gen}(\text{seed}_{\mathbf{A}})$
4: $\mathbf{S} \leftarrow \text{FrodoKEM.SampleMatrix}(\text{seed}_{\mathbf{E}}, n, \overline{n}, T_\chi, 1)$
5: $\mathbf{E} \leftarrow \text{FrodoKEM.SampleMatrix}(\text{seed}_{\mathbf{E}}, n, \overline{n}, T_\chi, 2)$
6: $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E} \pmod q$
7: $\dot{\mathbf{S}} \leftarrow \text{cSHAKE}(\text{seed}_{\mathbf{E}}, 16n\overline{n}, 0)$
8: $\ddot{\mathbf{S}} \leftarrow \mathbf{S} - \dot{\mathbf{S}}$
9: **return** public key $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B})$ and secret key $sk = (\mathbf{s}, \dot{\mathbf{S}}, \ddot{\mathbf{S}})$

Algorithm 16 FrodoKEM.Decaps with masking

Input: Ciphertext $c = (\mathbf{B}', \mathbf{C}, \mathbf{d}) \in \mathbb{Z}_q^{\overline{m} \times n} \times \mathbb{Z}_q^{\overline{m} \times \overline{n}} \times \{0,1\}^{\text{LEN}}$, secret key $sk = (\mathbf{s}, \dot{\mathbf{S}}, \ddot{\mathbf{S}}) \in \{0,1\}^{\text{LEN}} \times \mathbb{Z}_q^{n \times \overline{n}} \times \mathbb{Z}_q^{n \times \overline{n}}$ and public key $pk = (\text{seed}_{\mathbf{A}}, \mathbf{B}) \in \{0,1\}^{128} \times \mathbb{Z}_q^{n \times \overline{n}}$.

Output: Shared secret $\mathbf{ss} \in \{0,1\}^{\text{LEN}}$.

1: $\dot{\mathbf{M}} \leftarrow \mathbf{C} - \mathbf{B}'\dot{\mathbf{S}} \pmod q$
2: $\mathbf{M} \leftarrow \dot{\mathbf{M}} - \mathbf{B}'\ddot{\mathbf{S}} \pmod q$
3: $\text{seed}_{\mathbf{X}} \xleftarrow{\$} \mathcal{U}(\{0,1\})^{128}$
4: $\mathbf{X} \leftarrow \text{cSHAKE}(\text{seed}_{\mathbf{X}}, 16n\overline{n}, 0)$
5: $\dot{\mathbf{S}} \leftarrow \dot{\mathbf{S}} + \mathbf{X} \pmod q$
6: $\ddot{\mathbf{S}} \leftarrow \ddot{\mathbf{S}} - \mathbf{X} \pmod q$
7: $\mu' \leftarrow \text{FrodoKEM.Decode}(\mathbf{M})$
8: **return** $\text{FrodoKEM.CCA}(c, \mathbf{s}, \mu', pk)$

4.4 Summary and comparison

I shall now recap all of the different techniques explored in this chapter and give some indications on how they compare to each other.

- Shifting rows (or columns, depending on the multiplication) is a deterministic technique to break the symmetry exploited by the extend-and-prune attack. It is based on the simple idea of changing the order of operations every time a row (or column) of the secret matrix is processed, in such a way that the first element of each row (or column) is actually the first one to be processed only once. Despite being just a matter of indexing on paper, I will show in Section 5.6 how

tricky it gets when operands are not fully present in memory, as is the case for **A**. Given the attacks it aims at defending against, this countermeasure applies in the single-trace setting only, and is in fact useless against divide-and-conquer.

- As I mentioned in Section 2.3, attackers come up with power models to simplify the process of elaborating guesses to then compare against the power measurements. In the case of Frodo, however, knowing the device leaks according to a certain mathematical function can be leveraged in the defender's favour. In particular I showed that, in case such a function is the Hamming weight, Frodo can be modified so that all secret elements have the same Hamming weight, thanks to the fact that they are "small" modulo q. Needless to say, whenever the above assumption does not hold, the countermeasure fails too. When it does, however, it can be applied both in the single-trace setting and the multi-trace setting.

- Blinding is the process of breaking any relation between public information, hence known to the adversary, and operations involving secret values. This way, an adversary can no longer formulate guesses based on public knowledge. As before, it can be applied to both settings.

- Finally, masking aims at splitting the long-term secret matrix in two (or more) shares in such a way that leakage on each of them is independent of the secret, because they themselves are.

Each countermeasure applies in certain scenarios and under certain assumptions, therefore an overall comparison is hardly possible. In the single-trace setting, shifting rows (or columns) effectively prevents the most dangerous attacks because, as shown in Chapter 3, divide-and-conquer does not perform very well given the small number of traces available. Single-trace attacks were indeed more of a threat to the original FrodoKEP [BCD+16] rather than the newer FrodoKEM [NAB+17]. For the latter, an adversary would more likely target the long-term secret matrix **S**, rendering multi-trace countermeasures more relevant for real attacks. In this context, Hamming weight homogenisation, despite being applicable and interesting from a theoretical point of view, is based on very strong assumptions. Blinding is certainly preferable because the amount of randomness required is very limited while offering good protection against attacks targeting matrix multiplication in DECAPS. The drawback is that, since the secret is left untouched, attacks exploiting other parts of the protocol could easily circumvent it. This is the reason why I believe masking offers the highest degree of protec-

tion: the secret is split in two shares, which are independent from it if taken separately. The structure of FrodoKEM is such that shares never need to be recombined, while the constant refreshing makes any attack spanning multiple runs of the algorithm a lot more expensive. If one wished for an even stronger security guarantee, masking (with more than two shares) has been shown to be *provably* secure under certain assumptions [PR13], which is a feature none of the other countermeasures presented here have. Another, very important, metric of judgement among countermeasures is the performance penalty incurred once adopted. I defer this comparison to Section 5.6.

Chapter 5

Implementation of FrodoKEM-640

As I mentioned at the beginning of Chapter 3, the ARM Cortex-M0 processor was chosen as the target device mainly for the availability of the ELMO simulator, which made possible a more comprehensive and deeper analysis than if I worked on the physical device. If one had to choose where to implement FrodoKEM, however, the M0 would not be a natural or ideal pick. Because FrodoKEM is based on "ring-less" LWE, security is achieved through operations among fairly large matrices. Even generating and storing one such matrix are non-trivial tasks on a constrained device. Targeting bigger devices is therefore mandatory.

5.1 Overview

Performance of the implementation informs deployment in the real world. Constrained environments, included in the Internet of Things (IoT) framework, are especially challenging platforms. For instance, a naïve implementation of FrodoKEM-640, as provided by the original authors [NAB+17] on an ARM Cortex-A72, exceeds the resources available on such platforms, hence requiring particular care and analysis.

A popular choice for implementing cryptography on embedded devices is the ARM Cortex-M4, which has more resources then the aforementioned M0 but still sits in the IoT ecosystem of devices. The most delicate and resource intense operations are the expansion of the public pseudorandom matrix $\mathbf{A}$ from a seed and various matrix multiplications by smaller matrices. I exploit a combination of on-the-fly expansion of $\mathbf{A}$, originally proposed in the specifications [NAB+17], with a particular set of instructions available on my target platform through the Digital Signal Processing (DSP) extension.

These instructions fall into the Single Instruction, Multiple Data (SIMD) paradigm.

As the ARM Cortex-M4 is a 32-bit architecture, it can hold up to 32 bits in each internal register. Conveniently, FrodoKEM-640 is formed of matrices defined over $\mathbb{Z}_q$ with $q = 2^{15}$, which can be embedded in $\mathbb{Z}_{2^{16}}$ for convenience of computation. This implies that each element can be stored as a halfword in a register, and this is where SIMD instructions turn out to be most useful: when four values are correctly stored in the four halves of two registers, it is possible to operate on them in parallel with a single instruction. I will heavily use the `smlad` instruction, which multiplies corresponding halfwords from two registers, adds them together, accumulates the result to a third register and stores the final output in a fourth one, all in one instruction.

Furthermore, I analyse a suite of algorithms and memory layouts of matrices: with which function $\mathbf{A}$ is generated, either cSHAKE or AES, and whether it is involved in a multiplication as a left or as a right operand are all slightly different variants which require different optimisations.

Despite the carefully tailored optimisations, however, performance of matrix multiplications involving $\mathbf{A}$ is still dominated by its generation. A fine grained benchmark does indeed show that the greatest number of cycles is spent performing AES and cSHAKE. I analyse the rationale behind the choice of such cryptographically secure PRNGs, and conclude that they are over-conservative for the task of generating a public matrix from a public seed. I therefore suggest to use a different, non-cryptographic PRNG in order to speed up the generation of $\mathbf{A}$. I choose to implement FrodoKEM-640 using the PRNG xoshiro128** [BV18], which produces high quality pseudorandomness at a fraction of the cycle count. I show how to embed it in FrodoKEM-640 and benchmark it against the official two PRNGs.

5.1.1 Achieved performance

I improve multiplication to the right of $\mathbf{A}$ by up to 62.8%, and by up to 78.9% when multiplication is to the left of $\mathbf{A}$, compared to the reference [NAB+17]. Such results are achieved by carefully analysing the layout in memory of small matrices, and how portions of $\mathbf{A}$ are generated and stored. I coded tailored algorithms for each situation, while keeping SIMD instructions as a common design rationale for all the implementations. I also apply some of the algorithms to multiplication between "small" matrices, i.e. when $\mathbf{A}$ is not an operand: even in these cases I obtain improvements up to 78.9%, compared to the reference.

With faster multiplication routines, generating the matrix $\mathbf{A}$ becomes even more dominant than before. For instance, the number of cycles at a clock frequency of 24

MHz spent to generate **A** accounts for 86% of the whole execution of FrodoKEM-640: the remaining 14% can be as optimised as possible, but the difference will barely be noticeable anyway. I therefore suggest the usage of xoshiro128** to generate **A**, which suddenly becomes one of the cheapest operations in FrodoKEM-640: it gets reduced by up to 96%.

5.2 Preliminary notions

Section 2.4.4 already gave all the necessary details on FrodoKEM-640, thus here I limit myself to highlight those aspects which are particularly relevant for the implementations.

The ARM Cortex-M4 core is a popular choice for microcontroller usage and has become a representative platform to benchmark cryptographic application for usage in the IoT ([AJS16, HOKG18, KRS19, KRSS]). I target the ARM Cortex-M4 core as well to allow for easy comparison against previous applied cryptographic research.

The ARM Cortex-M4 in my setup is mounted on a STM32F407 board, equipped with 1 MB of flash and 192 KB of memory. The default clock frequency is 168 MHz. The board is equipped with a True Random Number Generator (TRNG) that derives entropy from analog noise. A Linear Feedback Shift Register (LFSR) is seeded with the noise and with a dedicated clock. Its output is stored in the RNG_DR register, which is read when randomness is needed. Checks to verify the absence of abnormalities in both the noise-derived seed and dedicated clock are present too.

5.2.1 Implementation details of FrodoKEM-640

The matrix **A** is the main bottleneck to any implementation of FrodoKEM-640 on embedded devices. Apart from being resource intensive to generate, it also simply does not fit in memory. The ARM Cortex-M4 platform is therefore ideal to test on-the-fly solutions, because it forces the generation of **A** in chunks.

In the official implementation of FrodoKEM-640, matrices are stored and operated upon as arrays. The convention used to linearise a two-dimensional structure like a matrix into one-dimensional arrays has a huge impact on performance on embedded devices. It changes which elements are adjacent in memory, affecting how they are loaded. There is no universal convention followed by matrices in FrodoKEM-640, rather it comes down to convenience on a case-by-case basis. Throughout this chapter, I use

teletype font to denote arrays obtained from matrices, e.g. a is the array corresponding to matrix **A**. Indexes start from 0, and I will make use of pointer arithmetic notation derived from the C programming language to make notation easier when handling arrays. If an $n \times n$ matrix **A** is stored row-wise in the vector a, for instance, then $\mathtt{a} + n$ denotes a pointer to the first element of the second row of **A**, as it lies n positions away from the base address (always pointing to the top-left element of a matrix: $A_{0,0}$).

Remark 5.1. It has to be noticed that there are ways of speeding up the computation of AES and cSHAKE by exploiting the fact that the inputs are partly fixed for each iteration. By reformatting how bitstrings are given, it is possible to compute and store parts of the internal state. For example, the plaintext of AES is mostly made of zeros, hence portions of the first and second rounds could be precomputed and reused at every iteration. Similarly, since $\text{seed}_{\mathbf{A}}$ is fixed, the first few absorptions of cSHAKE can be computed in advance. I do not explore these ideas further, as doing so would mean to enter in the details of how the two PRNGs are implemented for only very limited gains. I instead decided to use them as black boxes and focus on the operations in FrodoKEM-640.

5.2.2 Fast Arithmetic on an ARM Cortex-M4 Core

In recent works, the ARM Cortex-M4 core has been considered a representative platform when benchmarking post-quantum primitives targeting embedded applications [AJS16, HOKG18, KRS19, KRSS]. This platform has a word-size of 32 bits. The Cortex-M4 has specific instructions which can work on two half-words (of size 16 bits) in parallel following the *Single Instruction, Multiple Data* (SIMD) paradigm. The most relevant instructions I use in my implementations follow.

- `ldmia` loads multiple (up to eight) full-word values from consecutive memory into the corresponding amount of registers and, optionally, updates the pointer to the memory accordingly.
- `smlad` multiplies four half-word sized values stored in two word registers and adds them to an accumulator. More specifically,

  $$\texttt{smlad} \quad d, a, b, c$$

  computes

  $$d = \left(a \bmod 2^{16}\right) \cdot \left(b \bmod 2^{16}\right) + \left\lfloor \frac{a}{2^{16}} \right\rfloor \cdot \left\lfloor \frac{b}{2^{16}} \right\rfloor + c \bmod 2^{32}.$$

- `ldrh` and `strh` are useful for loading and storing half-word sized values from memory in one instruction, without needing to load a full-word combined with masking or shifting.
- `bfi` copies a bit field from one register into another one. This is useful for merging two half-word values for using the aforementioned SIMD instruction.

Given the amount of available registers these instructions allow the multiplication of two matrices five values at a time, in a pretty straightforward way. This is under the assumption that the left matrix is given in row-major order and the right matrix in column-major order such that they can both be accessed linearly. As will be outlined in Section 5.3 this is not always the case, hence the need for dedicated subroutines accessing non-adjacent portions of a linearised matrix.

5.3 FrodoKEM-640 Optimisations

Algorithms 3 and 4 differ in how they the fill the matrix **A**: cSHAKE returns one full row a the time while AES only generates 128-bit ciphertexts, corresponding to eight consecutive positions row-wise, hence allowing for a more flexible filling of **A**. Despite these differences, AES and cSHAKE do show a certain degree of similarity, enabling some common subroutines in ARM assembly to be reused in both cases. These will also turn out useful once I describe how I optimised matrix multiplications other than those by **A**, namely the ones between $n \times \overline{n}$ matrices in ENCAPS and DECAPS.

I opted for addressing different multiplications in different sections, rather than dividing the discussion between PRNGs. Therefore, Section 5.3.1 describes the common subroutines, which are used in Sections 5.3.2 and 5.3.3 that describe **AS** and $\mathbf{S}'\mathbf{A}$, respectively. I conclude with multiplications between small matrices in Section 5.3.4.

5.3.1 Common Subroutines

Both AES and cSHAKE can generate a full row of **A**, although in slightly different ways. This perfectly fits the case when **A** is the left operand in a multiplication, because it implies that some of its rows are fully stored in adjacent memory locations, and happens during KEYGEN only. Even more conveniently, **S** is stored column-wise, effectively opening the way to an inner product function based on SIMD instructions.

I call the first subroutine `ip_n`, as it computes the inner product between two vectors of length n, which I assume have their elements stored in adjacent locations in memory.

This will be the case for several matrices. The inner functionality and a description of all instructions follow.

ip_n

```
1   mov %[r], #0
2   ldmia %[s]!, {r0, r1, r2, r3, r4}
3   ldmia %[a]!, {r5, r6, r7, r8, r9}
4   smlad %[r], r0, r5, %[r]
5   smlad %[r], r1, r6, %[r]
6   smlad %[r], r2, r7, %[r]
7   smlad %[r], r3, r8, %[r]
8   smlad %[r], r4, r9, %[r]
9   (lines 2-8 are repeated 64 times)
```

1 The register holding the final result, named `%[r]`, is initialised to zero.

2-3 The instruction `ldmia` is used to load multiple registers at once. I load five registers, i.e. 20 bytes, exploiting the fact that all elements I am interested in are adjacent in memory. The exclamation mark after the address registers indicates that pointers, stored in registers `%[s]` and `%[a]`, are also updated, hence no other instructions will be needed in the next iteration.

4-8 Five SIMD instructions are everything I need to perform 10 multiplications and 10 additions (accumulations) on the output register `%[r]`.

9 Since the above lines compute over 10 elements at a time, I need to repeat them $n/10 = 64$ times.

The function `ip_n` crucially relies on the vectors to be multiplied to have adjacent positions in memory such that their addresses can be loaded once and then updated directly by the loading instruction. Unfortunately such an optimal scenario only happens when $\mathbf{AS}$ is performed during KEYGEN. Apart from there, $\mathbf{A}$ has to be generated during ENCAPS and for the re-encapsulation part of DECAPS. In these cases it is multiplied to the left by the matrix $\mathbf{S}'$, while still being generated row-wise for consistency and correctness. On the bright side, $\mathbf{S}'$ is generated row-wise.

The function I developed for the matrix multiplication $\mathbf{S}'\mathbf{A}$ is denoted `row_by_chunk` and is depicted in Figure 5.1. Four registers are loaded with eight consecutive elements in a column of $\mathbf{A}$ (I represent only four of them in Figure 5.1 for the sake of compactness).

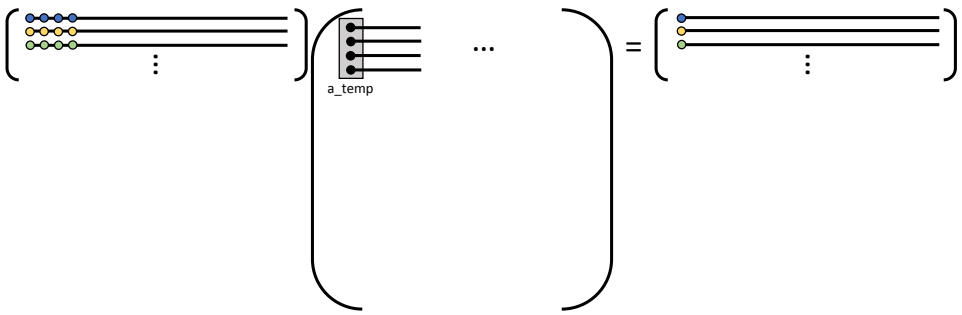


Figure 5.1: Visualisation of the `row_by_chunk` function. Circles refer to elements, where for compactness I depicted `a_temp` holding four elements, while in reality it holds eight. Lines show how elements are disposed in memory. Finally, the colour code simply highlights how multiplication by `a_temp` works.

As this procedure differs between cSHAKE and AES, I defer the details and the adopted instructions for later, while for now referring to those elements as the vector `a_temp`. Then, for every row of $\mathbf{S}'$, the corresponding eight elements are loaded, SIMD-multiplied with `a_temp` and the result is accumulated in the corresponding positions of the output matrix. The colour code in Figure 5.1 visualises them. Code and a line by line description follow.

`row_by_chunk`

```
1   mov r0, %[s]
2   mov r9, %[o]
3   ldmia r0, {r5,r6,r7,r8}
4   ldrh r10, [r9, #0]
5   smlad r10, r1, r5, r10
6   smlad r10, r2, r6, r10
7   smlad r10, r3, r7, r10
8   smlad r10, r4, r8, r10
9   strh r10, [r9, #0]
10  add r0, r0, #1280
11  (lines 3-10 are repeated 8 times)
```

1 The address of the matrix $\mathbf{S}'$ is held in `%[s]`.

2 The address of the matrix $\mathbf{B}'$ is held in `%[o]`. I use hard-coded offsets to access the correct position: each element in a column of $\mathbf{B}'$ is $2n = 1280$ bytes away from the top element in the same column.

3 Four registers are filled with eight elements of $\mathbf{S}'$. Note that I do not use the exclamation mark this time, because I manually modify the address to access non adjacent memory locations (line 10).

4 One element of $\mathbf{B}'$ is loaded: this is a 16-bit value, hence the instruction to load a halfword is used.

5-8 Four SIMD instructions are used to perform 8 multiplications and 8 additions (accumulations). I assume the elements of `a_temp` are stored in registers `r1` up to `r4`.

9 The partially updated element of $\mathbf{B}'$ is stored back in place.

10 The address of $\mathbf{S}'$ is shifted by $2n = 1280$ bytes, therefore it now points to the second row of $\mathbf{S}'$.

11 Lines 3 to 10 are repeated a total of $\overline{n} = 8$ times, hence partially updating a full column of $\mathbf{B}'$. The only caveat is that after four columns, the register `r9` holding the pointer to $\mathbf{B}'$ must be updated by $4 \cdot 2n = 5120$ bytes with an extra `add` instruction, otherwise the offset would exceed the maximum allowed by the architecture.

5.3.2 Optimising the Matrix Multiplication AS

Conveniently, the matrix $\mathbf{A}$ is the left operand and is stored row-wise. I can apply cSHAKE as specified in Algorithm 4 whenever I need a row, while in the case of AES I can fix the index i and run the j-loop from Algorithm 3. Once one or more rows are generated, I can simply run the `ip_n` subroutine directly.

In Algorithm 17 four rows of $\mathbf{A}$ are generated at-a-time, which works slightly different depending on whether AES or cSHAKE is being used. In the first case, an extra loop over the columns, eight by eight, is needed; while the latter PRNG generates full rows straight away. Next, I multiply each of the generated rows by all columns of $\mathbf{S}$ and accumulate the output vector. Note that `ip_n` returns a value, which is added in to `b`. Within the two if-branches, the same code as in Algorithms 3 and 4 is used.

5.3.3 Optimising the Matrix Multiplication S′A

In the setting of the matrix multiplication $\mathbf{S}'\mathbf{A}$, it is crucial to understand how $\mathbf{A}$ is placed in memory and how AES differs from cSHAKE. The `row_by_chunk` subroutine

Algorithm 17 AS multiplication

Input: Seed $\text{seed}_{\mathbf{A}} \in \{0,1\}^{128}$, output vector $\mathtt{b} \in \mathbb{Z}_q^{n\overline{n}}$ initialised with error matrix $\mathbf{E}$, and secret vector $\mathtt{s} \in \mathbb{Z}_q^{n\overline{n}}$.
Output: $\mathtt{b} \leftarrow \mathtt{b} + \mathbf{AS}$.
Parameters: $n = 640$, $\overline{n} = 8$, $q = 2^{15}$.

1: $\mathtt{a\_rows} \leftarrow \{0\}^{4n}$
2: **for** $0 \le i < n, i \leftarrow i + 4$ **do**
3: **for** $0 \le j < 4$ **do**
4: **if** PRNG = AES **then**
5: **for** $0 \le k < n, k \leftarrow k + 8$ **do**
6: $\mathbf{p} \leftarrow \langle i + j \rangle \,\|\, \langle k \rangle \,\|\, 0 \,\|\, \ldots \,\|\, 0 \in \{0,1\}^{128}$
7: $\mathtt{a\_rows} + j \cdot n + k \leftarrow \text{AES}_{\text{seed}_{\mathbf{A}}}(\mathbf{p})$
8: **else if** PRNG = cSHAKE **then**
9: $\mathtt{a\_rows} + j \cdot n \leftarrow \text{cSHAKE}(\text{seed}_{\mathbf{A}}, 16n, 2^8 + i + j)$
10: **for** $0 \le k < \overline{n}$ **do**
11: **for** $0 \le j < 4$ **do** ▷ Unrolled loop
12: $\mathtt{b}[k + (i + j)\overline{n}] \leftarrow \mathtt{b}[k + (i + j)\overline{n}] + \mathtt{ip\_n}(\mathtt{s} + k \cdot n, \mathtt{a\_rows} + j \cdot n)$

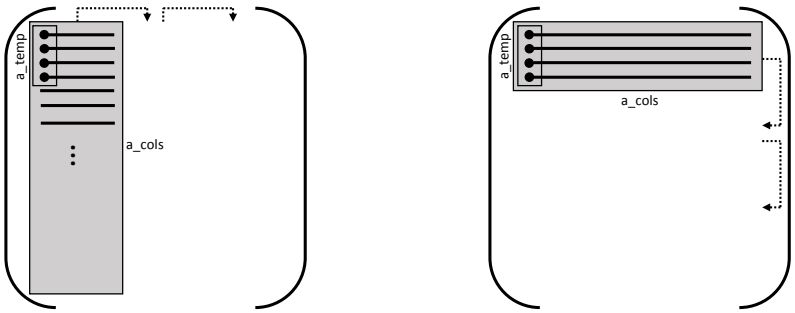


Figure 5.2: The matrix $\mathbf{A}$ as generated and accessed by AES (left) and cSHAKE (right). Dashed arrows show in which order `a_cols` moves to different portions of $\mathbf{A}$.

assumes eight consecutive elements in a column of $\mathbf{A}$, which do need to be adjacent in memory, are stored in the `a_temp` array. The latter is then multiplied by the corresponding portions of $\mathbf{S}'$. I denote the portion of $\mathbf{A}$ being generated each time as `a_cols`, which is the counterpart of `a_rows` from Section 5.3.2.

Figure 5.2 visualises the process in both the AES (left) and the cSHAKE (right) cases. In the former, an $n \times 8$ submatrix is generated by performing the whole i-loop with a fixed j in Algorithm 3. Performing the same operation with an incremented j yields the next submatrix, hence shifting `a_cols` to the right by 8 positions. Since cSHAKE can only generate full rows, `a_cols` is simply filled with the first eight of them

(only four are shown for compactness). Dashed arrows show in which direction `a_cols` moves in both cases. Note that Figure 5.2 represents only four elements in `a_temp` for compactness.

Once `a_cols` is generated and stored, values corresponding to `a_temp` have to be loaded to be used by `row_by_chunk`. This is where cSHAKE and AES differ: `a_temp` contains eight consecutive values in one column, and moves to the next column for the next iteration. Once there are no more columns available in `a_cols`, `a_temp` jumps to the subsequent eight rows. Such a wrap around happens after 8 columns for AES and n columns for cSHAKE. In particular, in the former case `a_temp` jumps more often, but the next eight rows are present in memory; in the latter `a_temp` covers the full eight rows present in memory, then `a_cols` has to be filled with the next eight rows. The following ARM assembly snippet illustrates how I dealt with such a discrepancy.

load_a_temp

```
1  _________aes_________|_________cshake________
2  ldrh r1, [%[a], #0]   | ldrh r1, [%[a], #0]
3  ldrh r5, [%[a], #16]  | ldrh r5, [%[a], #1280]
4  bfi r1, r5, #16, #16  | bfi r1, r5, #16, #16
5  ldrh r2, [%[a], #32]  | ldrh r2, [%[a], #2560]
6  ldrh r5, [%[a], #48]  | ldrh r5, [%[a], #3840]
7  bfi r2, r5, #16, #16  | bfi r2, r5, #16, #16
8                        | add %[a], %[a], #5120
9  ldrh r3, [%[a], #64]  | ldrh r3, [%[a], #0]
10 ldrh r5, [%[a], #80]  | ldrh r5, [%[a], #1280]
11 bfi r3, r5, #16, #16  | bfi r3, r5, #16, #16
12 ldrh r4, [%[a], #96]  | ldrh r4, [%[a], #2560]
13 ldrh r5, [%[a], #112] | ldrh r5, [%[a], #3840]
14 bfi r4, r5, #16, #16  | bfi r4, r5, #16, #16
```

The code snippet `load_a_temp` reports both AES (left) and cSHAKE (right) versions, with differences highlighted in red. Starting from the address stored in `%[a]`, *even* positions are loaded to the bottom half of each register, while *odd* ones are first loaded to the bottom half of the temporary register `r5` and subsequently moved to the top half of the designated register (`r1` up to `r4`) thanks to the `bfi` instruction. Such a procedure is common to both halves of the snippet.

What differs is the offset when loading values from `%[a]`. In the case of AES (left) the offset among values in the same column is a multiple of 16 bytes because `a_cols` is

Algorithm 18 $\mathbf{S}'\mathbf{A}$ multiplication

Input: Seed $\text{seed}_{\mathbf{A}} \in \{0,1\}^{128}$, output vector $\texttt{bp} \in \mathbb{Z}_q^{n\overline{n}}$ initialised with error matrix $\mathbf{E}'$, and secret vector $\texttt{sp} \in \mathbb{Z}_q^{n\overline{n}}$.
Output: $\texttt{bp} \leftarrow \texttt{bp} + \mathbf{S}'\mathbf{A}$.
Parameters: $n = 640$, $\overline{n} = 8$, $q = 2^{15}$.

```
1:  a_cols ← {0}^{8n}
2:  for 0 ≤ k < n, k ← k + 8 do
3:      if PRNG = AES then
4:          for 0 ≤ i < n do
5:              p ← i‖k‖0...0 ∈ {0,1}^128
6:              a_cols + 8·i ← AES_{seed_A}(p)
7:          for 0 ≤ i < n, i ← i + 8 do
8:              for 0 ≤ j < 8 do
9:                  a_temp ← load_a_temp_aes(a_cols + 8·i + j)
10:                 row_by_chunk(sp + i, a_temp, bp + k + j)
11:     else if PRNG = cSHAKE then
12:         for 0 ≤ j < 8 do                          ▷ Unrolled loop
13:             a_cols + j·n ← cSHAKE(seed_A, 16n, , 2^8 + k + j)
14:         for 0 ≤ i < n do
15:             a_temp ← load_a_temp_cshake(a_cols + i)
16:             row_by_chunk(sp + k, a_temp, bp + i)
```

an $n \times 8$ submatrix of $\mathbf{A}$. Instead, elements in the same column of `a_cols` when cSHAKE is used are $2n = 1280$ bytes apart from each other, hence offsets in the right half of the snippet are multiples of 1280 (cf. Figure 5.2). This introduces an extra complication: offsets are not allowed to exceed 4095, hence I have to spend an extra `add` instruction, highlighted in red, to make the address in `%[a]` point to the first position of the fourth row.

Algorithm 18 incorporates both AES and cSHAKE variants of the $\mathbf{S}'\mathbf{A}$ multiplication. I denoted by `load_a_temp_aes` and `load_a_temp_cshake` the two halves of the `load_a_temp` snippet. Once the appropriate values are loaded, the `row_by_chunk` function is applied. Note that it does not return any value, as the memory location of `bp` is an input, hence results are immediately stored back. Loop structures make sure that submatrices are traversed in the correct order.

Algorithm 19 $\mathbf{B}'\mathbf{S}$ multiplication

Input: Uninitialised output vector $\texttt{out} \in \mathbb{Z}_q^{\overline{n}n}$, vectors $\texttt{bp}, \texttt{s} \in \mathbb{Z}_q^{n\overline{n}}$ containing $\mathbf{B}'$ and $\mathbf{S}$, respectively.
Output: $\texttt{out} \leftarrow \mathbf{B}'\mathbf{S}$.
Parameters: $n = 640$, $\overline{n} = 8$, $q = 2^{15}$.

1: **for** $0 \le i < \overline{n}$ **do**
2: **for** $0 \le j < 8$ **do** ▷ Unrolled loop
3: $\texttt{out}[i \cdot \overline{n} + j] \leftarrow \texttt{ip\_n}(\texttt{bp} + in, \texttt{s} + j \cdot n)$

5.3.4 Small Matrix Multiplication Optimisations

Multiplications involving $\mathbf{A}$ are by far the most time consuming. There are however multiplications between smaller matrices ($\overline{n} \times n$ and vice-versa) which take a much smaller portion of the overall computation time, but can still benefit from some of the subroutines outlined in Section 5.3.1.

During DECAPS, the function `ip_n` can be used to efficiently perform $\mathbf{B}'\mathbf{S}$, since both matrices are conveniently stored in memory. Differently than before, they are fully present in memory, and the matrix $\mathbf{B}'$ is not initialised to an error matrix because it comes from the ciphertext. The result of multiplication is therefore saved in a different vector and is later subtracted from $\mathbf{C}$ using a different function.

Algorithm 19 shows how to use `ip_n` for the $\mathbf{B}'\mathbf{S}$ multiplication. Effectively I efficiently perform the inner product along the bigger dimension n, while the two smaller dimensions $\overline{n}$ are dealt with in a loop and in an unrolled fashion.

Similarly, I can compute the function $\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$ in both ENCAPS and DECAPS using the `row_by_chunk` function. In this case, $\mathbf{B}$ is an $n \times 8$ matrix stored row-wise, precisely like `a_cols` in the AES case because $\overline{n} = 8$. It is worth noticing that $\mathbf{B}$ is generated during KEYGEN, since it is part of the public key, and then used both in ENCAPS and DECAPS as the rightmost operand in matrix multiplications. Therefore a column-wise layout in memory would be more beneficial, but I do not explore this idea further because the impact on the overall computation time would be marginal. Algorithm 20 shows the pseudocode, which is exactly the same as in lines 6 to 9 of Algorithm 18.

I postpone the showcase of performances my optimisations achieve to Section 5.5, where a unified view on benchmarks, together with comparisons with other relevant works, will be given. In the coming section, instead, I focus my attention on how $\mathbf{A}$ is generated. So far I have been compliant with the design choices in the specification of

Algorithm 20 $\mathbf{S}'\mathbf{B}$ multiplication

Input: Output vector $\mathtt{v} \in \mathbb{Z}_q^{\overline{nn}}$ initialised with elements of $\mathbf{E}''$, vectors $\mathtt{b}, \mathtt{sp} \in \mathbb{Z}_q^{n\overline{n}}$ containing $\mathbf{B}$ and $\mathbf{S}'$, respectively.
Output: $\mathtt{v} \leftarrow \mathtt{v} + \mathbf{S}'\mathbf{A}$.
Parameters: $n = 640$, $\overline{n} = 8$, $q = 2^{15}$.

1: **for** $0 \leq i < n, i \leftarrow i + 8$ **do**
2: **for** $0 \leq j < \overline{n}$ **do**
3: `b_temp` $\leftarrow$ `load_a_temp_aes(b` $+ 8i + j$`)`
4: `row_by_chunk(sp` $+ i$`,b_temp,v` $+ j$`)`

FrodoKEM-640 [NAB+17], and optimised the usage of AES and cSHAKE inside matrix multiplications. I am about to offer an alternative greatly outperforming both.

5.4 Faster PRNG for A: xoshiro128**

Generating the large matrix **A** is typically problematic on embedded devices. More generally, the competitiveness of FrodoKEM-640 on any device is hampered by the need to deal with big matrices rather than small polynomials, as happens in its ring and module variants.

A step in the right direction has already been taken with the design choice of expanding true randomness with a PRNG. When it comes to generating large matrices, both randomness cost and performance improve. Setting aside the unavoidable cost of generating a seed from random, the next question is which PRNG to choose. As I pointed out several times, this choice impacts on how portions of a matrix are actually generated, thus on performance.

To remedy the performance penalty incurred by FrodoKEM-640 for generating **A**, I propose a third option for the PRNG by challenging the need for a cryptographically secure PRNG to generate a public matrix. The purpose of cryptographically secure PRNGs is to provide streams of pseudorandom numbers achieving some form of security. When the seed of the sequence is secret, an adversary learning the first k bits should not be able to predict the $(k+1)$th bit. If the internal state is compromised, reverting the computation upstream until the seed is disclosed should also be infeasible.

Therefore, the whole point of a cryptographically secure PRNG is that, as long as something inside remains secret, be it the seed or some internal state, an adversary cannot tell the difference between the output stream and a truly random sequence, nor recover anything that would allow backward reconstruction of the sequence. Trivially,

Algorithm 21 xoshiro128**

Input: State $s \in (\{0,1\}^{32})^4$.
Output: Pseudorandom number $r \in \{0,1\}^{32}$.
Parameters: None.

1: $r \leftarrow (((s[0] \cdot 5) << 7) \vee ((s[0] \cdot 5) >> 25)) \cdot 9$
2: $t \leftarrow s[1] << 9$
3: $s[2] \leftarrow s[2] \oplus s[0]$
4: $s[3] \leftarrow s[3] \oplus s[1]$
5: $s[1] \leftarrow s[1] \oplus s[2]$
6: $s[0] \leftarrow s[0] \oplus s[3]$
7: $s[2] \leftarrow s[2] \oplus t$
8: $s[3] \leftarrow (s[3] << 11) \vee (s[3] >> 21)$
9: **return** r

disclosing the secret reveals the deterministic nature of the sequence, hence making any security notion useless.

For the above reasons, I suggest the usage of a non-cryptographic PRNG over a secure one for the generation of the public matrix **A**, whose seed is part of the public key. This is advantageous because non-cryptographic PRNGs are usually faster and have smaller internal states, since they are only designed to achieve good statistical properties. The latter is still a stringent property of **A**, as clear patterns and other generic statistical weaknesses could potentially weaken the underlying LWE problem. I give more insights on this point in Section 5.4.2.1.

5.4.1 Description of xoshiro128**

I use xoshiro128** [BV18] as the designated PRNG for my implementation. There are several reasons behind this decision. As the name says, it has a 128 bit state, which is precisely the length of $\text{seed}_{\mathbf{A}}$. Moreover, the state is seen as an array of four 32 bit values, which matches the word size of the target architecture. However, larger versions achieving the same statistical properties are available too [BV18].

Statistical quality and performance are the main advantages of xoshiro128**. The design is inspired by the xorshift family of PRNGs [Mar03], which are linear functions and therefore are very fast but do not always pass all statistical tests. To fix this issue a *scrambler* is used, that is to say a non-linear layer to avoid linear dependencies in output. The authors of xoshiro128** use a sequence of multiplication-rotation-multiplication operations called a $**$ scrambler, hence the name xoshiro128**. One

word of the state is multiplied by constants of the form 2^s+1 and rotated. Constants are chosen to be efficiently implemented as one shift and one addition. Algorithm 21 shows the pseudocode of xoshiro128**, where << and >> indicate left and right rotations, respectively. Firstly the output of the current iteration r is computed, then the internal state `s` is updated.

Outputs of xoshiro128** achieve very good statistical quality. According to the original publication [BV18], xoshiro128** passes the BigCrush test from the TestU01 suite [LS07]. It is in fact one of the fastest doing so. On top of that, the authors designed a test to discover statistical bias in the Hamming weight of w-bit words generated by a PRNG, showing that xoshiro128** and closely related PRNGs succeed.

5.4.2 FrodoKEM-640 and xoshiro128**, Love at First Sight

I used xoshiro128** to generate **A** inside FrodoKEM-640. I left everything else untouched, including the fact that cSHAKE is used to generate $\text{seed}_{\mathbf{A}}$ and all secret and error matrices. For the latter, security of the PRNG is indeed mandatory, as the seeds are supposed to be kept secret. In the case of $\text{seed}_{\mathbf{A}}$, adopting a cryptographically secure PRNG fed with true randomness makes sure that tampering with the seed of **A** is not feasible, and that xoshiro128** is seeded correctly.

I opted for changing the order in which **A** is stored in memory: instead of having row values adjacent in memory, I generate **A** column-wise. This is motivated by the fact that the former convention is only convenient in KeyGen, while being problematic for Encaps and Decaps. This way, the situation is reversed and is overall beneficial as the latter two algorithms are supposed to run multiple times per key pair.

Column-wise layout in memory has another advantage: its symmetry with respect to the AES and cSHAKE generate **A**. I can therefore simply swap the subroutines described in Section 5.3.1 for the multiplications by **A**: `ip_n` can be used for $\mathbf{S}'\mathbf{A}$ and `row_by_chunk` can be used in $\mathbf{AS}$. The former returns a value, which is then added to the output matrix in the wrapping C code, but note that the latter stores results directly to memory. Therefore there is one caveat to take care of: `row_by_chunk` was originally designed to output $\mathbf{B}'$, which is saved in rows of length n, while here I want it to output $\mathbf{B}$, whose rows are of length $\overline{n} = 8$. A simple change of offsets in the loading and storing instructions is then enough to solve the issue.

I realised three implementations: a first one in portable C where **A** is fully pregenerated, an implementation in portable C where **A** is generated and multiplied on-the-fly which I use as reference, and finally the optimised implementation.

5.4.2.1 Security considerations

Swapping a cryptographic primitive for a construction which is not designed to meet any cryptographic notion is always problematic. It is a choice that does indeed sound like making the protocol relying on it insecure. However context is also very important and, in the scenario I am supporting the usage of xoshiro128**, several previous works suggest how this design choice is not catastrophic for security after all. The main reason behind this is that $\mathbf{A}$ does need to meet some statistical requirements, in such a way that using a cryptographically secure PRNG certainly checks all the boxes. The interesting question, however, is whether doing so is absolutely necessary to solve the problem of generating "good" matrices $\mathbf{A}$ or whether the same statistical properties (without secrecy properties, because $\mathbf{A}$ does not need any since it is public) can be achieved with other constructions as well.

In his technical report, Galbraith [Gal12] tackles precisely this research question by listing three properties that $\mathbf{A}$ needs to have in order for the resulting LWE instance to withstand known attacks. I hereby list the properties he pinpointed for completeness.

- As discussed in Section 2.4.3.1, the connection between LWE and hard problems over lattices heavily relies on $\mathbf{A}$, to the point that it finds a place in the notation itself of the lattice involved, namely $\Lambda_q(\mathbf{A})$. If a good basis of this lattice was easy to find via reductions from any base, clearly SVP and all related problems would be easy to solve. Therefore, the first property is that $\mathbf{A}$ should not allow finding good basis efficiently.

- Following the same notation of the scheme outlined in Section 2.4.4.2, it should be hard to solve the equation $\mathbf{C} = \mathbf{S}'\mathbf{A} + \mathbf{E}' \pmod q$ for $\mathbf{S}'$. Otherwise, an adversary can retrieve the message by computing $\mathbf{D} - \mathbf{S}'\mathbf{B} \pmod q$.

- It should be hard to find short vectors $\mathbf{v}$ such that $\mathbf{v}^\intercal\mathbf{A} = 0 \pmod q$. This is needed to avoid statistical weaknesses like repeated rows or other linear relationships between adjacent rows of $\mathbf{A}$.

Galbraith then proceeds in listing several PRNGs that might well fit the purpose, including some non-cryptographically secure ones like Linear Congruential Generator [Knu97, Section 3.2.2] and Mersenne Twister generator [MN98]. He then generalises his argument by stating that, and I quote, "We expect that much more lightweight pseudorandom word generators could be used in our application without loss of security." [Gal12, Section 6].

On top of Galbraith's analysis, there exist other instances in the literature where similar arguments where applied to use non-cryptographically secure PRNGs when generating public values or parameters. Ajtai [Ajt05], for the generation of a *public* parameter for his public-key cryptosystem, discussed how, clearly, using a non-cryptographic PRNG breaks down the security proof. However, since nothing about the parameter in question is kept secret, an adversary knows how to reconstruct the sequence so any non-random property could be exploited anyway. Perhaps more closely related to the Frodo's setting, Coron et al. [CMNT11a] use a non-cryptographic PRNG to expand a seed into several rational numbers which are part of the public key. Indeed they specify that, in their implementation, they used the PRNG in `glibc` [CMNT11b, Note 7].

5.5 Results and Comparison with Previous Works

Benchmarks are never straightforward to carry out, and interpreting their results to derive recommendations is an even more delicate task. In this section I collect all performance results of my implementations, as well as compare them with relevant previous works.

As I mentioned in Section 5.2, the default clock frequency of the target development board is 168 MHz. However when the main interest lies in how the implementation of a specific function performs, i.e. how many clock cycles the *operations* in that function take, then it is useful to run benchmarks at a lower clock frequency to compensate for memory accesses. If there are any inside the benchmarked function, the final number of clock cycles computed at a normal frequency might not be representative, as a good part of them might have been simply wasted by the function waiting for memory to retrieve the queried values. Memory is indeed notoriously slower than the microprocessor. On the other hand, not contextualising benchmarks at a lower frequency can be misleading and can lead to erroneous conclusions on the actual performance.

The STM32F407 development board offers a very neat and simple way of precisely computing cycle count, thanks to the Data Watchpoint and Trace (DWT) registers. I reset and read the DWT_CYCCNT register around functions to have a confident measure of how many cycles they take. I use the default 168 MHz clock frequency for benchmarking algorithms, and the lower 24 MHz when benchmarking cycle count of internal operations. This seems to be a popular choice in the literature [HOKG18, KRS19, KRSS], thus making comparisons fairer.

5.5.1 Relevant works for comparison

Howe et al. [HOKG18] recently implemented FrodoKEM-640 on the same target platform, yet did not use SIMD techniques. Secondly the PQM4 project [KRSS], which I partially used as the backbone of my implementation, is a unified framework to evaluate post-quantum candidates on an ARM Cortex-M4. These two works are my source of comparison.

Very recently, Kannwischer et al. [KRS19] implemented a plethora of schemes submitted to the NIST standardisation effort on an ARM Cortex-M4. The common denominator of all schemes is their use of polynomials in $\mathbb{Z}_{2^m}[x]$. Kannwischer et al. created a tool which automatically explores different divide-and-conquer multiplication approaches and generates assembly code for these different algorithms. The DSP assembly instructions are also adopted.

An implementation for the ARM Cortex-M4 of a lattice-based scheme using different moduli has also been implemented: Alkim et al. [AJS16] show how NewHope benefits from an optimised implementation.

I compare my implementations against the portable C implementation denoted as "optimised" in the official specifications [NAB+17]. Indeed, their "reference" implementation cannot possibly fit in my setup because $\mathbf{A}$ is fully generated, and it would occupy $2n^2 = 819{,}200$ bytes of memory, i.e. circa 820 KB against the 192 KB available. The "optimised" implementation, instead, generates and multiplies chunks of $\mathbf{A}$ on-the-fly. Since the latter is the only official implementation I can refer to for benchmarking purposes, I call it reference for the rest of the chapter.

I use the same implementation of cSHAKE as in the PQM4 framework [KRSS], which has been optimised in ARM assembly. Since AES is not part of the framework, I opted for the Schwabe and Stoffelen [SS16] implementation, also optimised for the ARM Cortex-M4 and used by Howe et al. [HOKG18] too. Finally, I deploy the implementation of xoshiro128** in C code, available at `http://xoshiro.di.unimi.it/xoshiro128starstar.c`.

5.5.2 Benchmarks at 168 MHz

I benchmark both the reference [NAB+17] and my optimised implementations for all three PRNGs. I set the clock frequency to the default 168 MHz, and execute the whole protocol 100 times, measuring cycle counts of KeyGen, Encaps and Decaps each time. Results are shown in Table 5.1.

Function	cSHAKE		AES		xoshiro128**
	Ref [NAB+17]	**This work**	Ref [NAB+17]	**This work**	**This work**
KEYGEN	99,762,353	88,288,589	105,892,559	95,593,272	14,205,132
ENCAPS	118,213,358	92,814,363	110,258,517	99,800,669	14,927,835
DECAPS	118,686,081	93,776,021	110,699,504	100,7471,40	15,731,086

Table 5.1: Cycle counts averaged over 100 executions and obtained at 168 MHz.

Overall, I improve the cycle counts of FrodoKEM-640 on the ARM Cortex-M4 by 18.4% when using cSHAKE, and by 9.4% when using AES. However, my optimised implementation using xoshiro128** is on average 84.3% faster than the above two, which is emblematic of how many cycles are spent in the generation of a publicly known matrix whose only goal is to look random.

I can fairly accurately estimate the time taken by FrodoKEM-640 in the three cases by dividing the total cycle count in Table 5.1 by $168 \cdot 10^6$ to obtain a time expressed in seconds. My optimised implementations take around 1.64 seconds when cSHAKE is adopted, 1.76 second with AES and 0.27 seconds with xoshiro128**.

5.5.3 Benchmarks at 24 MHz

For the second part of my results showcase, I downclock the microprosessor to 24 MHz, because I am interested in studying the number of cycles taken by my implementations to perform its operations, thus excluding time spent waiting for memory from the picture. On top of this, I also disable data cache: this forces the Schwabe and Stoffelen [SS16] implementation of AES to run in constant time, and slightly simplifies the flow of execution. Since memory is not a bottleneck at this clock frequency, keeping data in cache is not beneficial anyway.

Table 5.2 summarises the results of my implementations, in terms of clock cycles measured at 24 MHz and with data cache disabled. Several aspects are worth noticing. When it comes to the official PRNGs, my multiplication routines improve the reference ones by between 59.2% ($\mathbf{S}'\mathbf{A}$ with AES) and 78.9% ($\mathbf{S}'\mathbf{A}$ with cSHAKE). The most prominent fact about my version using xoshiro128**, instead, is the immensely better generation of $\mathbf{A}$: within the $\mathbf{AS}$ multiplication, it takes 91.1% less cycles than AES and 96.0% less than cSHAKE.

A further point of interest, which backs up my initial discussion on benchmarking at different clock frequencies, is in the performance in clock cycles of AES at 24 MHz in

Function	cSHAKE		AES		xoshiro128**
	Ref [NAB+17]	**This work**	Ref [NAB+17]	**This work**	**This work**
AS	90,053,180	78,068,108	48,764,566	38,449,619	10,811,476
Gen. **A**	73,612,819	71,948,659	32,315,402	32,327,961	2,867,639
Mult.	16,423,218	6,114,896	16,423,218	6,114,896	7,527,058
S′A	105,905,509	80,789,865	50,544,249	40,292,651	9,192,436
Gen. **A**	73,614,739	73,615,219	32,469,159	32,469,079	3,072,351
Mult.	35,955,867	7,578,258	17,240,744	7,041,221	6,115,378
KEYGEN	93,301,265	81,299,689	52,012,652	41,681,042	14,042,899
ENCAPS	111,616,496	86,255,368	56,255,552	45,758,155	14,657,940
DECAPS	112,084,038	87,212,153	56,684,122	46,720,217	15,456,163

Table 5.2: Cycle count of the reference and my optimised implementations, obtained at 24 MHz and with data cache disabled.

Table 5.2 when compared to that listed in Table 5.1 at 168 MHz. Despite AES consuming fewer cycles than cSHAKE when artificially clocked at 24 MHz, FrodoKEM-640 turns out to be slightly slower when using AES for the realistic 168 MHz frequency. This reversed behaviour is explained by the crucial role that memory access plays in AES, being based on tables, which is in turn almost neglected by counting cycles at 24 MHz.

I omitted matrix multiplications between small matrices from Table 5.2, as they do not depend on the PRNG. However, I improved also in this area: $\mathbf{S}'\mathbf{B}$ passed from 369,439 cycles to 111,103 (69.9% better), while $\mathbf{B}'\mathbf{S}$ from 410,222 to only 84,461 (79.4% better).

5.5.4 Comparison with Previous Works

As mentioned before, there are two relevant previous works I can compare my results against. The first one is the PQM4 project [KRSS], which I also used as a framework to embed and evaluate my code on the board. Secondly, Howe et al. [HOKG18] also recently proposed an implementation of FrodoKEM on the same microcontroller. Table 5.3 compares the cycle counts of KEYGEN, ENCAPS, DECAPS, as well as of some internal functions, across different PRNGs. All numbers, from all sources, have been obtained on the same board running at 24 MHz. Howe et al. [HOKG18] also disabled data cache since they used the same implementation of AES, thus the comparison holds. The impact of disabling data cache on the implementation based on cSHAKE is negligible, hence also the comparison with PQM4 [KRSS] is meaningful.

PRNG	Function	PQM4 [KRSS]	Howe et al. [HOKG18]	**This work**
cSHAKE	KeyGen	94,119,511	85,585,315	81,299,689
	Encaps	106,992,266	112,103,350	86,255,368
	Decaps	107,505,670	112,442,770	87,212,153
	AS		82,256,529	78,068,108
	$\mathbf{S'A}$		106,178,196	80,789,865
AES	KeyGen		44,603,160	41,681,042
	Encaps		47,742,966	45,758,155
	Decaps		48,051,929	46,720,217
	$\mathbf{AS}+\mathbf{E}$		41,308,745	38,449,619
	$\mathbf{S'A}+\mathbf{E'}$		41,833,535	40,292,651
xoshiro128**	KeyGen			14,042,899
	Encaps			14,657,940
	Decaps			15,456,163
	$\mathbf{AS}+\mathbf{E}$			10,811,476
	$\mathbf{S'A}+\mathbf{E'}$			9,192,436

Table 5.3: Cycle count of KeyGen, Encaps, Decaps and other functions as reported by previous works and compared to mine. Numbers were obtained on the same board, running at 24 MHz. Blank spaces refer to data not available.

5.5.4.1 A note on PQM4

At the time I performed experiments, PQM4 [KRSS] did not ship with a M4-specific implementation of FrodoKEM-640, hence performance benchmarks were done on the code that I used as a reference in this work. Moreover, only cSHAKE was implemented and benchmarked.

The attentive reader will notice a small discrepancy between the results of PQM4 in Table 5.3 and those of my reference implementation in Table 5.2. This is due to an update of the PQM4 framework (commit #23), in which the authors moved to a more recent version of GCC for ARM on Arch Linux, i.e. `arm-none-eabi-gcc` version 8.2.0. Since my setup is based on a different distribution I use `arm-none-eabi-gcc` version *7-2018-q2-update* as provided by ARM at `https://developer.arm.com/open-source/gnu-toolchain/gnu-rm`. For the sake of comparison, before said commit #23 PQM4 reported that KeyGen took 94,191,951 cycles, Encaps took 111,688,861 cycles and Decaps took 112,156,317 cycles. These numbers are indeed much closer to how the reference implementation [NAB+17] performs on my board, according to Table 5.2.

5.5.4.2 A note on Howe et al. [HOKG18]

Howe at al. [HOKG18] implemented both the AES and the cSHAKE flavours, including optimised routines in ARM assembly. Several differences with my work exist. First and foremost, they do not optimise the $\mathbf{S}'\mathbf{A}$ multiplication when cSHAKE is used, which makes my implementation the first of its kind and explains the bigger gap in the first section of Table 5.3. Note indeed that their ENCAPS and DECAPS cycle count when using cSHAKE is extremely close to the reference implementation in Table 5.2.

Secondly, they change the memory layout of the matrix $\mathbf{S}$ in the $\mathbf{AS}$ multiplication, for the sake of optimising loading patterns in that specific case. However, unless amended somewhere else in the code, this makes their implementation incompatible with the reference, while I chose to make $\mathbf{AS}$ and $\mathbf{S}'\mathbf{A}$ interchangeable with the reference.

Finally, they did not use SIMD instructions in their ARM assembly code, instead optimised for load/store operations. SIMD instructions do offer a speed-up, but require registers to be filled in a precise way, hence the design of the multiplication needs to be tailored around them. For instance, adjacent elements in memory can be loaded in multiple registers using the `ldmia` instruction, but then SIMD instructions can be used only if the two values contained in each register can be multiplied by values in the corresponding halves of other registers, and optionally also accumulated. Since they did not have such a (mild) restriction, they instead optimised for memory access patterns. Unfortunately, it is hard to give precise comparisons in terms of overall performance as their cycle counts lack contextualisation for different frequencies.

5.5.5 Comparison with other schemes

In order to capture how my work on Frodo sits in the panorama of lattice-based KEMs at large, I report a comparison with other candidates of the NIST standardisation effort too. This can be found in Table 5.4.

The results reported for FrodoKEM-640 are those from Table 5.1, which reports cycle count in the 168 MHz regime, in line with the other schemes too. I decided to include some representative schemes from the NIST standardisation effort, of which I picked the parameter set that matches the security level 1 from NIST, the one matching at least 128 bits of security, same as FrodoKEM-640. The chosen schemes are KYBER [SAB+19], NEWHOPE [PAA+19] and SABER [DKRV19]. The implementations from which I derived the numbers from, instead, are taken from the references in Table 5.4.

Scheme	KEYGEN	ENCAPS	DECAPS
FrodoKEM-640 cSHAKE	88,288,589	92,814,363	93,776,021
FrodoKEM-640 AES	95,593,272	99,800,669	100,7471,40
FrodoKEM-640 xoshiro128**	14,205,132	14,927,835	15,731,086
KYBER [ABCG20]	455,191	586,334	543,500
NEWHOPE [ABCG20]	578,890	858,982	806,300
SABER [KMRV18]	1,147,000	1,444,000	1,543,000

Table 5.4: Comparison between my work and other schemes. My results are those of Table 5.1.

As expected, all these schemes perform better than Frodo. This is due to the fact that they are all based on variants of the LWE problem which introduce further algebraic structure in order to improve the performance of implementations. Indeed KYBER and SABER are based on module lattices, while NEWHOPE is based on ideal lattices. See Section 2.4.3.4 for an introduction on this two types of lattices. As Table 5.4 shows, using xoshiro128** greatly contributes in bridging the performance gap between schemes based on plain LWE and those based on variants, however Frodo is still far from the latter. What is abundantly clear from this comparison is that the choice of Frodo over other schemes is really a trade-off between security and performance.

5.6 Cost of countermeasures

So far I have been presenting how I implemented, optimised and improved the *unprotected* version of FrodoKEM-640. It is interesting to evaluate how the countermeasures described in Chapter 4 affect performances. The extent to which a countermeasure is preferable over another one must indeed be informed both by its security guarantees and by whether the scenarios in which it is applicable match the scenarios in which it is applied, but also by potential penalties in terms of performance it incurs once it gets implemented.

In the remaining of this section I give implementation details for all countermeasures considered and defer an overall comparison of performances to Section 5.6.5.

5.6.1 Masking the accumulator

Masking the accumulator from Section 4.1.1 is fairly straightforward to implement on the ARM Cortex-M4. Indeed embedded devices usually have easy-to-access routines to generate randomness from, for instance, analog noise (see Section 5.2).

To apply this countermeasure it therefore suffices to draw the required amount of randomness before every matrix multiplication involving a secret matrix and subtracting it after the multiplication has occurred, as described in Algorithm 12.

5.6.2 Shifting rows against extend-and-prune

At the end of Chapter 3 I showed how the extend-and-prune strategy really fits how computations are carried out in FrodoKEM. Results, especially with a wider beam size, were indeed daunting. Luckily, Section 4.1 started on a much brighter note as I showed how a simple change in how matrix multiplication is computed suffices to structurally prevent extend-and-prune all together. As I briefly discussed there, the countermeasure is deterministic and can be implemented with almost no effort in pseudocode, as shown by Algorithm 13.

Despite its conceptual simplicity and apparent easiness of implementation, shifting rows is not as straightforward as it seems when it has to be combined with the optimisations described in Section 5.3, especially when $\mathbf{A}$ is an operand and is generated on-the-fly. A comprehensive list of the attack scenarios considered in Chapters 3 and 4 combined with implementation considerations follows.

- In case a matrix multiplication is computed between a secret matrix and a "small" matrix, i.e. of dimension $\overline{n} \times n$ or $n \times \overline{n}$, the shifting rows technique is easy to apply because all rows and columns are present in memory. A simple preprocessing of the two vectors the inner product is computed upon, then, suffices.

- In case $\mathbf{A}$ is an operand of the multiplication and is generated on-the-fly, different PRNGs affect the process in different ways. In more detail, when $\mathbf{A}$ is the left operand, e.g. in the computation of $\mathbf{AS}$ during KEYGEN, then a simple preprocessing as before suffices both when AES or cSHAKE are used, but not when xoshiro128** is used. This is due to the fact that I chose to generate $\mathbf{A}$ column-wise with xoshiro128**, hence rows never fully lie in memory. Dedicated routines are needed. Conversely, when $\mathbf{A}$ is the right operand, e.g. in the computation of $\mathbf{S}'\mathbf{A}$ during ENCAPS, then the shifting rows technique can be directly applied when

Algorithm 22 Shifting vectors in memory

Input: Vector $\mathtt{v} \in \mathbb{Z}_q^n$, shift $s < n$.
Output: $\mathtt{v}^* \leftarrow \mathtt{v} \gg s$.
Parameters: $n = 640$, $q = 2^{15}$.

1: **for** $0 \leq i < n$ **do**
2: $\quad \mathtt{v}^* + i \leftarrow \mathtt{v} + (i + n - s \pmod n)$
return $\mathtt{v}^*$

AES or xoshiro128** are used, but not when cSHAKE is used. Again this is because cSHAKE never generates full columns, which are required by this kind of matrix multiplication.

It must be noticed that the design choice of generating sub-matrices with AES pays off in flexibility when applying this countermeasure. In the two cases where optimisations are incompatible with the shifting rows countermeasure, I drop them and implement ad-hoc routines to still realise the countermeasure. In what follows, I describe the different implementation aspects categorised above, while postponing a discussion over achieved performances till the end of the section.

5.6.2.1 Implementation with vectors in memory

This is the simplest scenario, where the two vectors involved in the inner product computation are fully present in memory. In particular, two pointers to them exist and the elements of the vectors can be reached by adding an offset to the base pointer. This is the case for multiplication between "small" matrices, when **A** is the left operand and is generated by either AES or cSHAKE, or when **A** is the right operand and is generated by xoshiro128**.

Algorithm 22 shows the simple routine that is needed to shift a vector of length n by an index. The result is such that the new vector $\mathtt{v}^*$ has the desired layout, as described in Section 4.1.2, and has adjacent elements in memory. The value of the shift s is set accordingly. Algorithm 22 is run before any `ip_n` call, completing the implementation in this scenario. Note that, for the scope of the function, $\mathtt{v}^*$ occupies a different position in memory than v. The latter is then overwritten at the end of computation.

5.6.2.2 Implementation with vectors generated on-the-fly

When **A** is the right operand, hence its columns are needed, and is generated with AES, parts of the optimisations are still possible. Recall from Section 5.3, and as shown in

Algorithm 23 `load_a_temp_aes` with shifted columns

Input: Vector `a_cols` $\in \mathbb{Z}_q^{n\overline{n}}$, index i and shift $s < n$.
Output: `a_temp` $\leftarrow$ [`a_cols`$[i+n-s \pmod n], \ldots,$ `a_cols`$[i+n-s+7 \pmod n]$].
Parameters: $n = 640$, $\overline{n} = 8$, $q = 2^{15}$.

1: **for** $0 \leq k < 8$ **do**
2: `a_temp` $+ k \leftarrow$ `a_cols` $+ (i+n-s+k \pmod n)$
return `a_temp`

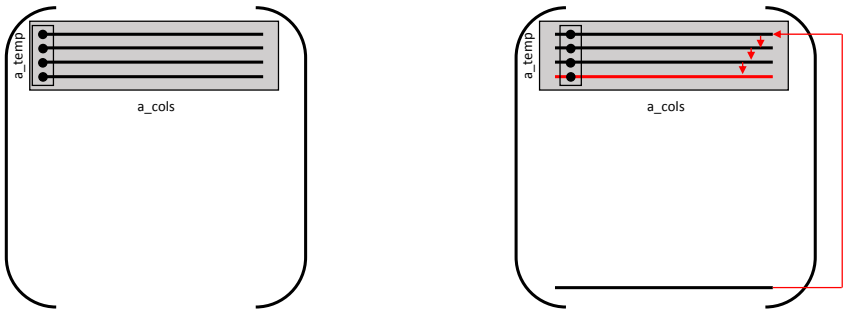


Figure 5.3: How the matrix **A** is generated and accessed when using cSHAKE to implement the shifting rows countermeasures.

Figure 5.2, that AES can generate columns whose elements are not adjacent in memory. The algorithm I used to pick the correct elements was `load_a_temp_aes`, hence by modifying it I can leave the actual multiplication routine untouched.

The modification is straightforward and is shown in Algorithm 23. The vector `a_cols`, see Figure 5.2, contains the elements of **A** generated so far. The index i in input corresponds to the index of the second loop of the AES branch of Algorithm 18, while $s < n$ is the amount by which columns are shifted, according to Section 4.1.2, and is equal to index j in the AES branch of Algorithm 18.

Algorithm 23 is then used to choose appropriate values from `a_cols`, while Algorithm 22 is used to modify the secret vector in the multiplication in such a way that computations are consistent with the new layouts.

When **A** is the right operand and is generated with cSHAKE a more involved procedure is required. I take the computation of $\mathbf{S}'\mathbf{A}$ as a running example to demonstrate the concepts behind this implementations. There, $\mathbf{S}'$ is fully present in memory in row-major order. On the contrary, **A** can only be generated row-wise, making it hard to shift columns.

Figure 5.3 shows how to traverse the matrix **A** in this scenario. The multiplication between the first eight positions (Figure 5.3 only shows four for compactness) of each

Algorithm 24 $\mathbf{S}'\mathbf{A}$ multiplication with shifting rows countermeasure

Input: Seed $\text{seed}_{\mathbf{A}} \in \{0,1\}^{128}$, output vector $\texttt{bp} \in \mathbb{Z}_q^{n\overline{n}}$ initialised with error matrix $\mathbf{E}'$, and secret vector $\texttt{sp} \in \mathbb{Z}_q^{n\overline{n}}$.
Output: $\texttt{bp} \leftarrow \texttt{bp} + \mathbf{S}'\mathbf{A}$.
Parameters: $n = 640$, $\overline{n} = 8$, $q = 2^{15}$.

```
1:  a_cols ← {0}^{8n}
2:  for 0 ≤ k < n, k ← k + 8 do
3:      for 0 ≤ j < 8 do                                   ▷ Unrolled loop
4:          a_cols + j·n ← cSHAKE(seed_A, 16n, 2^8 + k + j)
5:      for 0 ≤ i < n do
6:          if i > 0 then
7:              for 0 ≤ j < 6, 0 ≤ t < n do
8:                  a_cols + (7 − j)·n + t ← a_cols + (6 − j)·n + t
9:              a_cols ← cSHAKE(seed_A, 16n, 2^8 + (n − i + k (mod n)))
10:         a_temp ← load_a_temp_cshake(a_cols + i)
11:         row_by_chunk((sp ≫ i) + k, a_temp, bp + i)
```

row of $\mathbf{S}'$ and first eight elements of the first column of $\mathbf{A}$ is performed normally. This is depicted in the left panel of Figure 5.3. In the original algorithm, see Algorithm 18, the same eight positions of all rows of $\mathbf{S}'$ were then multiplied by the corresponding positions in the whole `a_cols`. Here, instead, the rows of $\mathbf{S}'$ are shifted with Algorithm 22 while `a_cols` is transformed as follows. The last row is discarded and all others are moved one position down. The first row is then replaced by the last row of $\mathbf{A}$, such that the first eight elements of the second column of `a_cols` correspond to the first eight elements of the second column of $\mathbf{A}$ when is shifted by one position. This process is shown in Figure 5.3, where the bottom row is discarded (hence red), all those above it are shifted down and the first one is filled with elements form the last row of $\mathbf{A}$. This process is repeated n times, once per column of $\mathbf{A}$. This is again repeated $n/8 = 80$ times, by using different portions of the rows of $\mathbf{S}'$. Algorithm 24 shows all the steps and is the equivalent of the cSHAKE branch of Algorithm 18.

Finally xoshiro128** poses a different kind of problem when $\mathbf{A}$ is the left operand. Since I adopted the convention to always generate $\mathbf{A}$ column-wise to favour ENCAPS and DECAPS over KEYGEN, the latter is the most affected algorithm. Moreover, the state of xoshiro128** is exactly the length of the seed to generate $\mathbf{A}$, hence no additional information containing which column to generate is present: the seed is fed into xoshiro128**, which then sequentially generates the full matrix from first to last col-

umn. Like cSHAKE, this is unfortunate when one has to shift rows.

The solution to this problem is conceptually simple but heavily impacts performance: every time a row is needed, the algorithm has to generate the whole matrix $\mathbf{A}$ and only retain those elements belonging to it while discarding the others. In my actual implementation, I leveraged a time-memory trade off: instead of keeping elements of one row only, I keep those of eight. Once rows are generated and stored adjacently in memory, they can be shifted and multiplied (with SIMD instructions) with shifted columns of $\mathbf{S}$.

5.6.3 Hamming weight homogenisation

The Hamming weight homogenisation is straightforward to apply, by means of the formulae described in Section 4.2. In particular, the long-term secret $\mathbf{S}$ can be transformed in KEYGEN and stored in the secret key in that form. The ephemeral secret $\mathbf{S}'$ needs to be transformed once in ENCAPS and once in DECAPS, for a total of three homogenisations. All multiplication routines need to change accordingly to apply the correction factors from Section 4.2: this is easy even in multiplications by $\mathbf{A}$, because the factors can be computed incrementally as $\mathbf{A}$ gets generated, and applied at the end.

5.6.4 Blinding and masking

I have already mentioned in Section 5.6.1 how generating randomness on embedded devices is not particularly cumbersome, thanks to dedicated routines. This already solves the biggest problem usually associated with blinding and masking, i.e. their hunger for randomness. It turns out that implementing Algorithms 14 to 16 and integrating these countermeasures into FrodoKEM-640 is made particularly easy thanks to the possibility of reusing some functions which are already part of FrodoKEM, which I list next.

- $\text{seed}_{\mathbf{X}} \xleftarrow{\$} \mathcal{U}(\{0,1\})^{128}$ and $\mathbf{X} \leftarrow \text{cSHAKE}(\text{seed}_{\mathbf{X}})$ from Algorithm 14 can be implemented in the exact same way seed generation and expansion of the matrix $\mathbf{S}'$ are implemented. The algorithms responsible for these two functions are called `randombytes` and `cshake`, respectively.

- $\mathbf{M} \leftarrow \mathbf{M_x} - \mathbf{XS} \pmod q$ from Algorithm 14 and $\mathbf{M} \leftarrow \mathbf{M} - \mathbf{B}'\ddot{\mathbf{S}} \pmod q$ from Algorithm 16 are implemented with the same function that carries out the equivalent operation in the unprotected version of DECAPS, called `frodo_mul_bs`.

Implementation	Function	cSHAKE	AES	xoshiro128**
Unprotected	KeyGen	88,288,589	95,593,272	14,205,132
	Encaps	92,814,363	99,800,669	14,927,835
	Decaps	93,776,021	100,747,140	15,731,086
Masking Accumulator	KeyGen	89,708,474	95,849,902	14,470,564
	Encaps	93,070,942	99,999,392	15,219,012
	Decaps	94,018,015	100,946,093	16,002,015
Shifting	KeyGen	99,672,887	106,638,677	179,531,964
	Encaps	2,926,299,959	110,560,054	26,914,166
	Decaps	3,180,899,262	113,971,892	30,335,248
Hamming	KeyGen	93,805,284	100,083,325	15,757,593
	Encaps	94,880,533	101,562,513	20,208,843
	Decaps	95,849,749	102,531,755	21,016,805
Blinding	KeyGen	88,318,863	95,480,794	14,204,353
	Encaps	92,844,883	99,688,430	14,926,810
	Decaps	94,355,746	101,209,561	16,289,292
Masking	KeyGen	89,907,891	96,035,898	14,700,598
	Encaps	92,815,678	99,731,847	14,931,691
	Decaps	94,056,792	100,979,193	16,090,799

Table 5.5: Cycle count of KeyGen, Encaps, Decaps of unprotected and protected implementations. Results are averaged over 100 executions and obtained at 168 MHz.

- Finally, I implemented the many steps which add or subtract matrices with dimensions $n \times \overline{n}$ anew, and as they are component-wise operations the implementation is trivial.

5.6.5 Performance results

It is now time to compare the performances of all countermeasures proposed against that of the unprotected implementation. The metric I chose to carry out the comparison is cycle count when the board runs at 168 MHz, i.e. default frequency. I believe this is preferable over downclocking to 24 MHz, as done in Section 5.5 to compare with other works, because grasping the differences in performance is more intuitive.

Table 5.5 shows, for each alternative implementation and each PRNG used to expand **A**, an average based on 100 executions of KeyGen, Encaps and Decaps. Note that experiments reported in the cSHAKE and xoshiro128** columns have negligible standard deviations, while experiments reported in the AES column have a standard

deviation approximately equals to 295,500. This is enough to justify why some countermeasures seem faster than the unprotected version, while in reality the distributions of clock cycles are very similar. Several conclusions can be drawn from these numbers.

When thwarting extend-and-prune, a trade-off exists: shifting rows (or columns) was presented as a deterministic countermeasure that breaks the structure extend-and-prune relies upon. It is indeed very easy to implement by changing how **A** is accessed and traversed, however the implicit assumption that **A** must fully reside in memory is required. As described in Section 5.6.2, when this is not the case accessing the required portions of **A** is extremely costly because those portions need to be re-generated several times to accommodate how the multiplication routine changes. This fact clearly reflects in Table 5.5, where shifting rows has devastating consequences on performances, especially on cSHAKE, which can only generate full rows, and on xoshiro128**, for which the index of column to be generated cannot be given. For these reasons, shifting is preferable only on devices with enough memory to store **A**, for instance the ARM Cortex-A72 used in the original specifications, which is powerful enough to cache **A**.

Hamming weight homogenisation affects each PRNG slightly differently, because the corrections factors that are needed to maintain correctness are computed depending on how **A** is generated. The impact is rather small, where the apparent bigger impact on xoshiro128** is simply due to its speed: the overhead is a greater percentage of the total cycle count than in the case of cSHAKE and AES because many cycles are spent to generate **A** in the latter two case, and are therefore unrelated to the countermeasure itself. All in all, the strong assumption it requires, the security arguments reported in Section 4.2 and the small yet noticeable overhead make this countermeasure a poor choice.

A very different discussion applies to blinding and masking. These countermeasures are meant to protect the long-term secret matrix **S**, which already excludes any modification to the ENCAPS algorithm and, more in general, to any routines involving **A**. This is a huge difference compared to previous countermeasures, because generating and computing over **A** is the main bottleneck, hence masking and blinding have a big advantage in terms of performance. Effectively, this means that they are not comparable with other countermeasures proposed here. Performances between blinding and masking are extremely similar, with differences being mainly due to experimental errors rather than actual run-time differences. Therefore, any preferences on one or the other should be conducted, in this case, solely on their different security guarantees.

Chapter 6

Conclusions

The journey I have narrated in this thesis started off from the three questions I listed in Chapter 1. According to Question 3, the practicality of post-quantum schemes needs to be assessed by implementing and benchmarking them on several different platforms. In the original submission [NAB+17], FrodoKEM was implemented on a x64 Intel processor and on a microprocessor of the ARM Cortex-A family, to which I added my implementation of FrodoKEM-640 on the ARM Cortex-M4 in Chapter 5. Questions 1 and 2, instead, inspired me to evaluate the security of post-quantum implementations. This is the reason why I explored several attack techniques in Chapter 3 and suggested some countermeasures, both novel and from the literature, in Chapter 4.

6.1 Summary of contributions

All in all, there are several points which are worth summarising.

- The most valuable lesson from Chapter 3 comes from Section 3.5, which contains an apparently counterintuitive concept. Parameters to instantiate a scheme are usually derived from the analysis of applicable attacks: a scheme is as secure as the efficiency of the best one against it. The spectrum of attacks considered to derive parameters of Frodo are mostly mathematical attacks against LWE [NAB+17]. Parameters are set in such a way that the their running time is deemed infeasible. Clearly, including side-channel attacks in the analysis is cumbersome: their effectiveness depends on a great deal of unknowns, most of which cannot even be estimated in general as the same attack might perform drastically differently whether algorithms are implemented on one platform over an-

other. Nonetheless, the story told in Chapter 3 points out that one can gain some general insights from side-channel analysis too. For the attacks analysed in this thesis, larger (square) matrices offer a wider surface for adversaries to retrieve sensitive information from leakage. This could potentially inform the recommendation to prefer small parameter sets (or non-square matrices, as is the case in EMBLEM [SPL+17]) on target platforms where power analysis is a concern, like IoT devices.

- Chapter 3 is also a neat confirmation that many tools and techniques developed in the side-channel literature still apply to post-quantum algorithms, and even become especially relevant. This is the case, for instance, for the extend-and-prune technique which found a fertile ground for application in operations like matrix multiplications involving secret elements.

- This opens the way for the study of countermeasures developed for post-quantum algorithms, which I addressed in Chapter 4. In this domain, there is space for both ad-hoc countermeasures, like those in Sections 4.1 and 4.2, and well-known techniques that just need to be adapted to the specific computation, as in Section 4.3. Despite there being a common ground between the two kinds, which is what made possible the comparison in Section 4.4, there simply cannot be any clear winner: leakage profile, assumptions on the adversarial powers and goals, number of traces available are all different dimensions that yield different trade-offs in terms of whether a countermeasure is better than another one both in terms of security and practicality.

- Yet another interesting point raised in Chapter 3, which comes as another confirmation of the relevance of side-channel theory to the realm of post-quantum cryptography, is the trend of improvement among Sections 3.3.2 and 3.3.3 and Section 3.4. At each step of such a ladder of better and better attacks, an adversary would drop assumptions about the leakage profile and would infer the information needed on the "shape" of the leakage from the device itself through experiments. In terms of the quantities defined in Section 2.3 (which come from the reference textbook on side-channel analysis [MOP07]), the adversary is exploiting more and more portions of the information leakage called P_{exp}, culminating in the close to perfect exploitation capability by means of the extend-and-prune technique.

- Picking a specific device is not only important for side-channel security, but also

for implementation and optimisation of a cryptoscheme. In Chapter 5, I showed how instructions native to the ARM Cortex-M4 turned out to be very beneficial for the performance of FrodoKEM-640. Despite many careful optimisations, however, little could be done to avoid the fact that **A** simply does not fit in the memory of such a small device, therefore had to be generated on-the-fly, with all the drawbacks this brings. For this reason, I put forward the idea of using a different, more efficient, subroutine to generate it in Section 5.4. I believe this incompatibility with the published version of FrodoKEM [NAB+17] is necessary to speed up FrodoKEM on embedded devices. The only apparent drawback of using xoshiro128** instead of cSHAKE or AES in the generation of **A** is that it has not been designed with security in mind. However, as I argued in Section 5.4, the only requirement for **A** is not to exhibit statistical pitfalls to avoid weakening the underlying LWE problem. Clearly, in the proof of security it is no longer possible to swap xoshiro128** with ideal functions but the pseudorandomness of **A** should follow from an heuristic argument, extensively corroborated by the original publication of xoshiro128** [BV18].

Finally, it must be noted that although the whole thesis rotates around Frodo, many arguments do not depend on its specifics, but rather focus on aspects of the underlying structure of LWE with "small" secrets. For this reason, similar conclusions can be drawn for other post-quantum schemes sharing the same structure as Frodo. In what follows, I will detail how my work is relevant for other KEMs submitted to the first round of the NIST post-quantum standardisation effort.

6.2 Relevance to other schemes

In order for some of the arguments discussed in this thesis to apply, a candidate scheme needs to be based on the LWE problem with "small" secrets, i.e. where the latter get values from a much smaller range than the whole $\mathbb{Z}_q$. Matrix multiplications are the real core operations to which the majority of conclusions apply. This means that many variants based on LWE are not suitable: just to name the two most famous ones, Ring-LWE is based on a particular kind of polynomial multiplication which allows, after a suitable transformation, component-wise multiplication and addition of coefficients; Module-LWE uses very small matrices (whose dimensions range between 2 and 5) of the same polynomials as Ring-LWE. For these reasons, the following list only contains schemes based on variants of LWE based on operations over $\mathbb{Z}_q$.

EMBLEM [SPL+17] is a KEM whose security is based on a variant of the LWE problem according to which secret matrices are drawn from the uniform distribution over a small interval of the kind $[-B, B]$, where B is much smaller than the modulus q. The latter is a power of two and error matrices follow a discrete Gaussian distribution. The structure of EMBLEM closely resembles that of FrodoKEM and, in fact, all matrix multiplications contained in KEYGEN, ENCAPS and DECAPS are laid out precisely in the same order and involve equivalent matrices. This makes all conclusions drawn in Chapters 3 and 4 applicable to EMBLEM as well. For the sake of a more quantitative comparison, the parameter B, that effectively determines the cardinality of the range from which secret matrices are drawn, is only set to 1 or 2, depending on the parameter set. This means that there are only three or five, respectively, possible values for each position of any secret matrix. In addition to that, the public matrix is $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ for $m = 1003$ and $n = 770$ or $m = 832$ and $n = 611$, respectively. Differently than FrodoKEM, $\mathbf{A}$ is not symmetric: a comparison is still possible in terms of a projected success rate of extend-and-prune from Section 3.4 by taking the minimum of its dimensions as the number of traces for each parameter set. In terms of Figure 3.7, therefore, both parameter sets of EMBLEM would sit very low on the y-axis and somewhere in the middle of the x-axis. According to the conclusions drawn in Section 3.5, this would make EMBLEM easier to attack than FrodoKEM, given the very low number of candidates compared to the number of traces per candidate available to an adversary. On the implementation-side, the size of $\mathbf{A}$ would still be problematic on a device equipped with the ARM Cortex-M4. EMBLEM, however, uses a modulus of $q = 2^{24}$ for both parameter sets, therefore a single element would be 8 bits bigger than a halfword in a 32-bit microprocessor, defeating the benefits of SIMD instructions.

KCL [ZJGS17] is a family of algorithms among which there is a Key Exchange Protocol based on LWE. In this case, the comparison is straightforward as the authors of KCL explicitly acknowledge being inspired by FrodoKEP [BCD+16], to the point that its parameters are extremely similar to those reported in Table 2.3 and the distributions of secret and error matrices to those reported in Table 2.4. This means, in particular, that all conclusions derived in the single-trace analysis from Chapter 3 are particularly relevant to KCL as it does not have a KEM counterpart.

LOTUS [PHAM17] is a KEM based on the hardness of the LWE problem with secret

and error matrices drawn from the discrete Gaussian distribution. The latter is set to have mean equal to 0 and standard deviation equal to 3.0, meaning that the returned range is comparable to that of NIST2. The dimension of $\mathbf{A}$, which is a square matrix in this case, ranges between 576, i.e. similar to CCS2, to 832, i.e. similar to CCS4. A crucial difference between LOTUS and FrodoKEM, however, prevents an equally meaningful comparison than with EMBLEM. During ENCAPS, LOTUS does not use an ephemeral secret *matrix* but an ephemeral secret *vector*. As a result, encapsulated keys are smaller, only matrix–vector and vector–matrix multiplications take place during ENCAPS and during DECAPS, but the long-term secret matrix needs to be up to 27 times bigger than the secret matrix in the NIST2 parameter set of FrodoKEM (the secret matrix in the biggest parameter set in LOTUS is of dimension 832×256, against the 976×8 of NIST2). Consequently, single-trace analysis from Chapter 3 only applies to the KEYGEN algorithm of LOTUS. Also, despite SIMD optimisations from Chapter 5 being possible thanks to the modulo of $q = 2^{13}$, the secret matrix of the smallest parameter set is circa 120 KB in size, i.e. almost filling the total memory of 192 KB. Given that the error matrix to compute the public key is as big, different on-the-fly computations than for FrodoKEM would be needed.

6.3 A note on the evolution of Frodo

By the time I started to write my thesis, the NIST post-quantum standardisation effort took a step forward and passed from the first round, which this thesis is based upon, to the second round. Submitters had the opportunity to make small modifications to their schemes in case they were admitted to the second round. This is the case for FrodoKEM. In order to fill the gap between my work, which is based on the version from the first round, I now list the main modifications it underwent from the update dated March 30^{th} 2019 and how they affect the results in my thesis.

- A new parameter set with $n = 1344$ was added to target a higher security level. Clearly this has no effect on the thesis.

- The standard deviation of FrodoKEM-640 was raised to account for a mistake in a security proof. While changing slightly the numbers in Table 2.2, results are unaffected.

- Matrices drawn from χ are no longer generated with multiple calls to the PRNG with different domain separators, but with a single call that outputs more bits. Effectively, this change makes the two versions of FrodoKEM incompatible as different matrices are produced from the same seed. Apart from the values inside the matrices, however, every argument in the thesis still holds.

- cSHAKE was replaced by SHAKE to reduce the number of calls to the underlying primitive KECCAK. As above, this change makes versions incompatible, however nothing changes dramatically because cSHAKE is simply a wrapper to KECCAK with pre-defined parameters, and so is SHAKE. Again, conceptual notions from this thesis continue to hold.

- Several changes in the security proofs were made, none of which have a noticeable effect on anything written in this thesis, as I am more concerned about implementation aspects.

6.4 Future directions

My work on lattice-based cryptography, particularly on LWE-based schemes as exemplified by Frodo but applied to other schemes too in Section 6.2, has approached two main topics: performance and side-channel security. There are several natural aspects that could definitely be pushed forward in both areas. I discussed several ideas to improve performance, such as using platform-specific implementation paradigms (e.g. SIMD) to boost performance on certain devices and using non-cryptographic PRNGs to expand public matrices. Natural next questions to research would be which other platforms can benefit from tailored implementations and what other ideas can be deployed to speed up even further LWE-based schemes, which are definitely behind in terms of performance compared to other schemes. A similar line of reasoning can be applied to side-channel analysis: the more attack techniques and countermeasure the community explores, the more confident everyone can be that real-world implementations of lattice-based schemes are secure in the wild.

I acknowledge that the above might sound a bit vague, as indeed they are future directions which are particularly specific to lattice-based cryptography, nor to post-quantum cryptography in general. All cryptographic primitives should be made faster and more secure in a side-channel sense therefore, despite obvious, the above research questions are definitely valid and worth pursuing. However, there is another direction

in the area of side-channel analysis of lattice-based cryptography which has always fascinated me and is, in a sense, rather unique. It has to do with the idea that leakage could somehow be accounted for in the underlying lattice structure. This is not a new idea in side-channel analysis, since it is tied in with the concept of leakage-resilient cryptography [DP08], for which a designer models leakage as a function on certain inputs and shows the scheme secure even in presence of such leakage. For lattice schemes, there seems to be very natural ways to "weaken" secret keys while still retaining an acceptable level of security [GKPV10]. This fact can be used to build schemes where leakage of some form is allowed, therefore being still resistant when it occurs in practice. On the other side of the spectrum, understanding better how leakage can be exploited to weaken the underlying lattice problem is useful to analyse more sophisticated attacks where leakage, thought of as partial information on the secret material, is actually used as a pre-processing step for lattice reduction algorithms. If this was the case, even an unsatisfactory side-channel attack like the one described in Section 3.3 might still turn useful to leverage the extra information gained, which is not enough by itself to break the scheme, as a base to mount a lattice attack.

This is a research question which I personally find really intriguing and that, I believe, has great potential. Despite I was not able to pursue it directly during my PhD, I am very happy that the paper and the code on which Chapter 3 is based upon were recently acknowledged by Dachman-Soled, Ducas, Gong and Rossi [DSDGR20], who did manage to make a significant contribution in this direction by providing a framework in which various leakage instances are analysed and integrated in a framework to cryptanalyse lattice-based schemes.

Appendix A

ARM assembly code for inner product

ARM Assembly inner product

```
.syntax unified
.text
.thumb

.global Vec_Mult

.func Vec_Mult
Vec_Mult:

 push {r1-r7}
  @Load and prepare the data
  @ i->0
  movs r4, #0
  @ number limit->address limit
  lsls r2, #1
 loop:
  @Load first[i]
  ldrh r5,[r0,r4]
  @Load second[i]
  ldrh r6,[r1,r4]
  @Multiply
  muls r5,r6
```

```
        @Add
        adds r3,r3,r5
        @Update i as address
        adds r4,r4,#2
        @Compare with limit
        cmp r4,r2
       bne loop
        @Return Value
        mov r0,r3
        pop {r1-r7}
        bx lr
       .endfunc
```

Appendix B

List of papers outside the scope of this thesis

I published and released several works beside those that contributed to the content of this thesis. A list of all other papers of mine follow.

- I published a conference paper [BM16] and its journal version [BMM17], together with Guido Bertoni and Maria Chiara Molteni, called "A Methodology for the Characterisation of Leakages in Combinatorial Logic". We explored particular algebraic structures to model the propagation of glitches in combinatorial circuits and how they could affect security. These papers deal with a very different area of cryptography than post-quantum cryptography, therefore they are not included in this thesis. Moreover, the material published in those two papers was collected in M.Sc. thesis.
- Daniel Martin and I released a paper in which we analysed some aspects of key rank [MM18]. For the same reason as above, this paper is not included here.
- James Howe, Ayesha Khalid, Francesco Regazzoni, Elisabeth Oswald and I published the paper "Fault Attack Countermeasures for Error Samplers in Lattice-Based Cryptography" at the IEEE ISCAS conference [HKM$^+$19]. Despite being relevant to the main theme of my thesis, my contribution to the content and write-up has been fairly marginal, which is why I have not included it here. This and other relevant papers from the literature will be discussed in more details in Chapter 6.

References

[ABCG20] Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-M4 optimizations for {R,M}lwe schemes. Cryptology ePrint Archive, Report 2020/012, 2020. `http://eprint.iacr.org/2020/012`.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, Heidelberg, August 2009.

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC 1997, pages 284–293. Association for Computing Machinery, 1997.

[AD17] Martin R. Albrecht and Amit Deo. Large modulus ring-LWE ≥ module-LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 267–296. Springer, Heidelberg, December 2017.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange—a new hope. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 327–343. USENIX Association, 2016.

[AJS16] Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. NewHope on ARM Cortex-M. In *Security, Privacy, and Applied Cryptography Engineering*, pages 332–349. Springer International Publishing, 2016.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *In Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 99–108. ACM, 1996.

[Ajt05] Miklós Ajtai. Representing hard lattices with O(n Log n) bits. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 94Ű103. Association for Computing Machinery, 2005.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Mathematical Cryptology*, 9(3):169–203, 2015.

[ATT+18] Aydin Aysu, Youssef Tobah, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018*, 2018.

[BB03] David Brumley and Dan Boneh. Remote timing attacks are practical. In *USENIX Security 2003*. USENIX Association, August 2003.

[BBK16] Nina Bindel, Johannes Buchmann, and Juliane Kramer. Lattice-based signature schemes and their sensitivity to fault attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2016.

[BCD+16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1006–1018. ACM Press, October 2016.

[BCG+14] Johannes Buchmann, Daniel Cabarcas, Florian Göpfert, Andreas Hülsing, and Patrick Weiden. Discrete ziggurat: A time-memory trade-off for sampling from a Gaussian distribution over the integers. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 402–417. Springer, Heidelberg, August 2014.

[BCLv19] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[BCNS15] Joppe Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with

errors problem. In *IEEE Symposium on Security and Privacy*, volume 2015, 2015.

[Ben80] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22(5):563–591, 1980.

[BFM+18a] Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam. Assessing the feasibility of single trace power analysis of Frodo. In Carlos Cid and Michael J. Jacobson Jr:, editors, *SAC 2018*, volume 11349 of *LNCS*, pages 216–234. Springer, Heidelberg, August 2018.

[BFM+18b] Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam. Fly, you fool! Faster Frodo for the ARM Cortex-M4. Cryptology ePrint Archive, Report 2018/1116, 2018. `https://eprint.iacr.org/2018/1116/20181120:031936`.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. 6(3), 2014.

[BHLY16] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 323–345. Springer, Heidelberg, August 2016.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Krawczyk [Kra98], pages 1–12.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013.

[BM16] Guido Bertoni and Marco Martinoli. A methodology for the characterisation of leakages in combinatorial logic. In Claude Carlet, Anwar M. Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 363–382. Springer International Publishing, 2016.

[BMM17] Guido Bertoni, Marco Martinoli, and Maria Chiara Molteni. A methodology for the characterisation of leakages in combinatorial logic. *Journal of Hardware and Systems Security*, 1(3):269–281, 2017.

[Bra16] Matt Braithwaite. Experimenting with post-quantum cryptography. `https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html`, 2016. Accessed:30/01/2020.

[BV18] David Blackman and Sebastiano Vigna. Scrambled linear pseudorandom number generators. *CoRR*, abs/1805.01407, 2018.

[CG13] Ran Canetti and Juan A. Garay, editors. *CRYPTO 2013, Part I*, volume 8042 of *LNCS*. Springer, Heidelberg, August 2013.

[Cha15] Subrata Chakraborty. Generating discrete analogues of continuous probability distributions-a survey of methods and constructions. *Journal of Statistical Distributions and Application*, 2(6), 2015.

[CJL+16] L. Chen, S. Jordan, Y.-K. Liu, D. Moody, R. Peralta, R. Perlner, and D. Smith-Tone. Report on post-quantum cryptography. NISTIR 8105, National Institute of Standards and Technology, 2016. `http://csrc.nist.gov/publications/drafts/nistir-8105/nistir_8105_draft.pdf`.

[CK14] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In *CARDIS'13*, volume 8419 of *LNCS*, pages 253–270. Springer, 2014.

[CMNT11a] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In Rogaway [Rog11], pages 487–504.

[CMNT11b] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. Cryptology ePrint Archive, Report 2011/441, 2011. `http://eprint.iacr.org/2011/441`.

[CMV+15] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. C. Cheung, D. Pao, and I. Verbauwhede. High-speed polynomial multiplication architecture for Ring-LWE and SHE cryptosystems. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(1):157–166, 2015.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Heidelberg, August 2003.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Krawczyk [Kra98], pages 13–25.

[dCRVV15] R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Efficient software implementation of Ring-LWE encryption. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 339–344, 2015.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Canetti and Garay [CG13], pages 40–56.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DKRV19] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. SABER. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography in the standard model. Cryptology ePrint Archive, Report 2008/240, 2008. `http://eprint.iacr.org/2008/240`.

[DS13] M. H. Devoret and R. J. Schoelkopf. Superconducting circuits for quantum information: an outlook. *Science*, 339(6124):1169–1174, 2013.

[DSDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. Cryptology ePrint Archive, Report 2020/292, 2020. `http://eprint.iacr.org/2020/292`.

[EFGT17a] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Loop-abort faults on lattice-based Fiat-Shamir and Hash-and-Sign

signatures. In *Selected Areas in Cryptography – SAC 2016*, pages 140–158. Springer International Publishing, 2017.

[EFGT17b] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-Channel Attacks on BLISS Lattice-Based Signatures. In *2017 ACM Conference on Computer and Communications Security (CCS 2017)*, pages 1857–1874, Dallas, TX, United States, October 2017. ACM.

[EFGT17c] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1857–1874. ACM Press, October / November 2017.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Fey82] Richard P. Feynman. Simulating physics with computers. *Internaional Journal of Theoretical Physics*, 21(6-7), 1982.

[Fey86] Richard P. Feynman. Quantum mechanical computers. *Foundations of Physics*, 16(6), 1986.

[Gal12] Steven D. Galbraith. Space-efficient variants of cryptosystems based on learning with errors, 2012. `https://www.math.auckland.ac.nz/~sgal018/compact-LWE.pdf`.

[Gar95] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly Media, Inc., 1995.

[gch] Milestones: Invention of public-key cryptography, 1969–1975. `https://ethw.org/Milestones:Invention_of_Public-key_Cryptography,_1969_-_1975`. Accessed: 26/10/2020.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178. ACM, 2009.

[GGH97a] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Eliminating decryption errors in the Ajtai-Dwork cryptosystem. In Kaliski Jr. [Kal97], pages 105–111.

[GGH97b] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Kaliski Jr. [Kal97], pages 112–131.

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. *Innovations in Computer Science*, January 2010.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM, 1996.

[Gün90] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37. Springer, Heidelberg, April 1990.

[HKM+19] J. Howe, A. Khalid, M. Martinoli, F. Regazzoni, and E. Oswald. Fault attack countermeasures for error samplers in lattice-based cryptography. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019.

[HOKG18] James Howe, Tobias Oder, Markus Krausz, and Tim Güneysu. Standard lattice-based key encapsulation on embedded devices. 2018(3):372–393, 2018. https://tches.iacr.org/index.php/TCHES/article/view/7279.

[HPS96] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: a new high speed public key cryptosystem. Preprint, presented at the rump session of Crypto '96, 1996. https://www.ntru.org/f/hps96.pdf.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe P. Buhler, editor, *Algorithmic Number Theory*, pages 267–288. Springer Berlin Heidelberg, 1998.

[HS98] Jeffrey Hoffstein and Joseph H. Silverman. Implementation notes fro NTRU PKCS multiple transmission. NTRU Cryptosystems technical report n6v1, 1998. https://www.ntru.org/f/tr/tr006v1.pdf.

[JF11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, pages 19–34. Springer Berlin Heidelberg, 2011.

[Kal97] Burton S. Kaliski Jr., editor. *CRYPTO'97*, volume 1294 of *LNCS*. Springer, Heidelberg, August 1997.

[KBF+15] J. Kelly, R. Barends, A. G. Fowler, A. Megrant, E. Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, I.-C. Hoi, C. Neill, P. J. J. O'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, A. N. Cleland, and John M. Martinis. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature*, 519:66–69, 2015.

[KH18] Suhri Kim and Seokhie Hong. Single trace analysis on constant time CDT sampler and its countermeasure. *Applied Sciences*, 8, 2018.

[KjHCP16] John M. Kelsey, Shu jen H. Chang, and Ray A. Perlner. SHA-3 derived functions: cSHAKE, KMAC, TupleHash and ParallelHash. Technical report, 2016.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.

[KMRV18] Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede. Saber on ARM CCA-secure module lattice-based key encapsulation on ARM. Cryptology ePrint Archive, Report 2018/682, 2018. `https://eprint.iacr.org/2018/682`.

[Knu97] Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, August 1996.

[Kra98] Hugo Krawczyk, editor. *CRYPTO'98*, volume 1462 of *LNCS*. Springer, Heidelberg, August 1998.

[KRR+18] A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Constant-time discrete gaussian sampling. *IEEE Transactions on Computers*, 67(11):1561–1571, 2018.

[KRS19] Matthias J. Kannwischer, Joost Rijneveld, and Peter Schwabe. Faster multiplication in $\mathbb{Z}_{2^m}[x]$ on Cortex-M4 to speed up NIST PQC candidates. In Robert H. Deng, Valérie Gauthier-Umaña, Martin Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 281–301. Springer, Heidelberg, June 2019.

[KRSS] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. `https://github.com/mupq/pqm4`.

[KY76] D. Knuth and A. Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.

[KY11] Abdel Alim Kamal and Amr Youssef. Fault analysis of the NTRUEncrypt cryptosystem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E94.A(4):1156–1158, 2011.

[Lor63] Edward N. Lorenz. *Deterministic Nonperiodic Flow*, volume 20, pages 130–148. 1963.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.

[LS07] Pierre L'Ecuyer and Richard Simard. TestU01: A C library for empirical testing of random number generators. *ACM Trans. Math. Softw.*, 33(4):22:1–22:40, August 2007.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Heidelberg, April 2012.

[Man80] Yuri Manin. *Computable and Uncomputable*. Sovetskoye Radio, 1980.

[Mar03] George Marsaglia. Xorshift RNGs. *Journal of Statistical Software, Articles*, 8(14):1–6, 2003.

[McE78] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. 44, 1978.

[Mer79] Ralph C. Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, Stanford University, 1979.

[MI88] Tsutomu Matsumoto and Hideki Imai. Public quadratic polynominal-tuples for efficient signature-verification and message-encryption. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 419–453. Springer, Heidelberg, May 1988.

[Mil82] Frank Miller. *Telegraphic Code to Insure Privacy and Secrecy in the Transmission of Telegrams*. C.M. Cornwell, 1882.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 417–426. Springer, Heidelberg, August 1986.

[MM18] Daniel P. Martin and Marco Martinoli. A note on key rank. Cryptology ePrint Archive, Report 2018/614, 2018. `https://eprint.iacr.org/2018/614`.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3Ű30, 1998.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., 2007.

[MOW] David McCann, Elisabeth Oswald, and Carolyn Whitnall. Implementation of ELMO. `https://github.com/bristol-sca/ELMO`. Accessed: 27-11-2017.

[MOW17] David McCann, Elisabeth Oswald, and Carolyn Whitnall. Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017*, pages 199–216. USENIX Association, August 2017.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In Canetti and Garay [CG13], pages 21–39.

[MSPA08] Mark D. McDonnell, Nigel G. Stocks, Charles E. M. Pearce, and Derek Abbott. *Stochastic Resonance – From Suprathreshold Stochastic Resonance to Stochastic Signal Quantization*. Cambridge University Press, 2008.

[MT00] George Marsaglia and Wai Tsang. The ziggurat method for generating random variables. *Journal of Statistical Software*, 05, 2000.

[MVM09] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Advanced Encryption Standard*. Alpha Press, 2009.

[MW17] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 455–485. Springer, Heidelberg, August 2017.

[NAB+17] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2017. Available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[NAB+19] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2019. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[Nat16a] National Institute of Standards and Technology. Post-quantum cryptography standardization. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf, 2016.

[Nat16b] National Institute of Standards and Technology. Report on post-quantum cryptography. https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf, 2016.

[Nat19] National Institute of Standards and Technology. Status report on the first round of the NIST post-quantum cryptography standardization process. https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf, 2019.

[Nat20] National Institute of Standards and Technology. Status report on the second round of the NIST post-quantum cryptography standardization process. https://csrc.nist.gov/publications/detail/nistir/8309/final, 2020.

[OG17] Tobias Oder and Tim Güneysu. Implementing the NewHope-simple key exchange on low-cost FPGAs. In Tanja Lange and Orr Dunkelman, editors, *LATINCRYPT 2017*, volume 11368 of *LNCS*, pages 128–142. Springer, Heidelberg, September 2017.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure masked Ring-LWE implementations. *IACR TCHES*, 2018(1):142–174, 2018. https://tches.iacr.org/index.php/TCHES/article/view/836.

[PAA+19] Thomas Poppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, Douglas Stebila, Martin R. Albrecht, Emmanuela Orsini, Valery Osheter, Kenneth G. Paterson, Guy

Peer, and Nigel P. Smart. NewHope. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[Pes16] Peter Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In *Progress in Cryptology – INDOCRYPT 2016*, pages 153–170. Springer International Publishing, 2016.

[PHAM17] Le Trieu Phong, Takuya Hayashi, Yoshinori Aono, and Shiho Moriai. LOTUS. Technical report, National Institute of Standards and Technology, 2017. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 513–533. Springer, Heidelberg, September 2017.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Heidelberg, May 2013.

[PRR19] Thomas Prest, Thomas Ricosset, , and Mélissa Rossi. Simple, fast and constant-time gaussian sampling over the integers for falcon. Second PQC Standardization Conference, 2019. https://csrc.nist.gov/CSRC/media/Events/Second-PQC-Standardization-Conference/documents/accepted-papers/rossi-simple-fast-constant.pdf.

[RdCR+16] Oscar Reparaz, Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Additively homomorphic Ring-LWE masking. In Tsuyoshi Takagi, editor, *7th Post-Quantum Cryptography (PQC'16)*, volume 9606 of *LNCS*, pages 233–244. Springer, Heidelberg, 2016.

[Red77] Dabbala R. Reddy. Speech understanding systems. summary of results of the five-year research effort at Carnegie-Mellon University. Technical report, 1977.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Rog11] Phillip Rogaway, editor. *CRYPTO 2011*, volume 6841 of *LNCS*. Springer, Heidelberg, August 2011.

[RRdC+16] Oscar Reparaz, Sujoy Sinha Roy, Ruan de Clercq, Frederik Vercauteren, and Ingrid Verbauwhede. Masking Ring-LWE. *J. Cryptographic Engineering*, 6(2):139–153, 2016.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[SAB+19] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[Sha49] Claude Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 1949.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS'94. IEEE Computer Society, 1994.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5), 1997.

[Sho01] Victor Shoup. A proposal for an ISO Standard for public key encryption(version 2.1). `http://www.shoup.net/papers/iso-2_1.pdf`, 2001. [Online; accessed 01-October-2018].

[Sil99] Joseph H. Silverman. High-speed multiplication of (truncated) polynomials. NTRU Cryptosystems technical report n10v1, 1999. `https://www.ntru.org/f/tr/tr010v1.pdf`.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 443–461. Springer, Heidelberg, April 2009.

[SPL+17] Minhye Seo, Jong Hwan Park, Dong Hoon Lee, Suhri Kim, and Seung-Joon Lee. EMBLEM and R.EMBLEM. Technical report, National Institute of Standards and Technology, 2017. Available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[SS16] Peter Schwabe and Ko Stoffelen. All the AES you need on Cortex-M3 and M4. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 180–194. Springer, Heidelberg, August 2016.

[SW06a] Joseph H. Silverman and William Whyte. Timing attacks on NTRUEncrypt via variation in the number of hash calls. NTRU Cryptosystems technical report n19v1, 2006. `https://www.ntru.org/f/tr/tr019v1.pdf`.

[SW06b] Joseph H. Silverman and William Whyte. Timing attacks on NTRUEncrypt via variation in the number of hash calls. In *Topics in Cryptology – CT-RSA 2007*, pages 208–224. Springer Berlin Heidelberg, 2006.

[VPR19] Felipe Valencia, Ilia Polian, and Francesco Regazzoni. Fault sensitivity analysis of lattice-based post-quantum cryptographic components. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 107–123. Springer International Publishing, 2019.

[Wel] D. Welch. Thumbulator. `https://github.com/dwelch67/thumbulator.git/`.

[WO11] Carolyn Whitnall and Elisabeth Oswald. A comprehensive evaluation of mutual information analysis using a fair evaluation framework. In Rogaway [Rog11], pages 316–334.

[ZCH+19] Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, William Whyte, John M. Schanck, Andreas Hulsing, Joost Rijneveld, Peter Schwabe, and Oussama Danba. NTRUEncrypt. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[ZJGS17] Yunlei Zhao, Zhengzhong Jin, Boru Gong, and Guangye Sui. KCL (pka OKCN/AKCN/CNKE). Technical report, National Institute of Standards and Technology, 2017. Available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[ZSS19] Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. Facct: Fast, compact, and constant-time discrete gaussian sampler over integers. *IEEE Transactions on Computers*, PP:1–1, 09 2019.